Professional Expertise Distilled

DevOps with Windows Server 2016

Obtain enterprise agility and continuous delivery by implementing DevOps with Windows Server 2016

Ritesh Modi

[PACKT] PUBLISHING enterprise professional expertise distilled

Table of Contents

1
Introducing DevOps

“Change is the only constant in life” is something I always kept hearing from childhood till now and would continue to hear for time immemorial. I never understood the change – the school remained same, the curriculum was same for years, home was same, friends were more or less the same. However, once I joined my first software company it immediately struck me that yes, “Change is the only constant!!“. Change is inevitable for any product or service and it gets many fold amplified if its related to software product, system or service.

Software development is a complex undertaking comprising of multiple processes, tools and involves people from different departments. They all need to come together and work in a cohesive manner. With so many variabilities, the risks are high while delivering to the end customers. One small omission or misconfiguration and the application might come down crashing. This book is about adopting and implementing practices that reduces this risk considerably and ensure that high quality software can be delivered to customer again and again. This chapter is about explaining how DevOps brings people, process, culture and technology together for successful delivery of software services to customer effectively and efficiently. It is focused on these DevOps theory and concepts. Rest of the chapters will focus on realization of these concepts through practical examples using Microsoft Windows 2016 and Visual studio team services. This chapter will help understand – what is DevOps, why is devops needed, what problems are resolved by DevOps, what are its constituents, principles and practices. It will set up the context that would continue with you as a reader for rest all the chapters in this book.

Before getting into details about DevOps let's understand problems faced by software companies that are addressed by DevOps.

Software deliver challenges

There are inherent challenges when engaged into activity of software delivery. It involves multiple people with different skills using different tools and technologies with multiple different processes. It is not easy to being all these together in a cohesive manner. Some of these challenges are discussed next and later we will see how eventually DevOps helps in overcoming them.

Changing resistance

Organizations works within the realms of political, economic and social backdrops and they have to constantly adapt themselves to the continuous changing environment. Economic changes might introduce increase in competition in terms of price, quality of products and services, change marketing strategies, mergers and acquisitions. Political environment introduces changes in legislation which has impact on rules and regulation for an enterprise. The tax system, international trade policies are few impactful areas. Society decides which products and services are acceptable or preferred while others are discarded. Customers demand change on a constant basis. Their need and requirements change often and they manifestation of same in the systems they are using. Organizations not adept in handling changes in their delivery processes, resisting change in their product and features eventually find themselves outdated and irrelevant. These organizations are not responsive to change. In short, there is ever changing environment and companies perish if they do not change along with them.

Rigid processes

Software organization with traditional mindset releases their products and services on yearly or multiyear basis. The software development lifecycle is long and operations do not have many changes to deploy and maintain. Customers are demanding more but they have to wait till next release from company. The company is not interested or does not have capability to release change faster. Meanwhile, if other company is providing more and better features, customer change their loyalty and start using them. The formal organization starts losing customers, has lower revenues and eventually fades away.

Isolated teams

There are generally multiple teams behind any system or services provided to the customer. Typically, there is development team and an operations team. Development team is

responsible for developing and testing the system while Operations team is responsible for managing and maintaining the system on production. Operations team provide post deployment services to the customer. These two teams have different skills, experiences, mindset and working culture. The charter of development team is to develop newer features and upgrading existing features. They constantly produce code and want to see them on production. However, operations team is not comfortable with frequent changes. Stability of the existing environment is most important to them. There is a constant conflict between these two teams.

There is no or very little collaboration and commination between these teams. Development team often provides code artifacts to operations team for deploying them on production without providing any help in making them understand the change. Operations team is not comfortable deploying the new changes since neither they are aware of the kind of changes coming in as part of new release nor have confidence in deploying the software. There is no proper hand off between development and operations team. More often the deployments fail on production and Operations team have to spend sleepless nights to ensure that either the current deployment is fixed or rolled back to previous working release. Both Development and operations team are working in Silos. Development team does not treat operations team as equivalent to themselves and looked down. Operations team has no role to play in entire software development lifecycle while Dev team has no role to play in operations.

Monolithic design and deployments

Companies have adopted the practice of releasing their software systems to market on yearly or multiyear basis. The software development lifecycle was long and operations did not have many changes to deploy and maintain. Development goes on for multiple months before testing start on them. The flow is linear and the approach is waterfall where next stage in software development lifecycle happens only when the prior stage is completed or nearing completion. Deployment is one giant exercise deploying multiple artifacts on multiple server based on documented procedures. Such practices have lot of inherent problems. There are a lot of features and configuration steps for large application and all needs to be done in order on multiple servers. Deploying a huge application is risky and fails when a small step is missed while deployment. It generally taken weeks to deploy a system like this on production. This type of application lifecycle is also known as true waterfall approach,

Manual execution

Enterprise software development also do not employ proper automation in their application lifecycle management. Developers tend to check-in code only after a week, the testing is manual, configuration of environment and system is manual, documentation is either missing or very heavy comprising of hundreds of pages. Operations team follows the

provided documentation to deploy the system manually on production. More often this results in large downtime on production because of missing smaller steps in deployment. Eventually customers are not satisfied with the company provided services. Also, this introduced human dependencies within the organization. If a person leaves the organization, so leaves the knowledge with him and a new person has to struggle significantly to gain the same level of expertise and knowledge.

Lack of innovation

Companies starts losing out to competition when they cannot be flexible to meet their customer expectation with newer and upgraded products and services. The end result is falling in profits and eventually closing down. We have had many examples in the past for the same. Companies are not innovating newer products and services, they do not update their existing portfolio of products and services and provide linear customer satisfaction. Sooner or later, other competitive companies take over the major market share while it strives to exist.

What is DevOps

Today, there is no consensus in industry regarding the definition of DevOps. Every organization has formulated their own definition of DevOps and have tried to implement it. They have their own perspective and think they have implemented DevOps if they have automation in place, configuration management is enabled, using agile processes or any other combination.

DevOps is about the delivery mechanism of software systems. It is about bringing people together, making them collaborate and communicate, working together towards common goal and vision. It is about taking joint responsibility, accountability and ownership. It is about implementing processes that fosters collaboration and service mindset. It enables delivery mechanism that brings agility and flexibility within the organization. Contrary to popular belief, DevOps is not about tools, technology, automation. These

are enablers that help in collaboration, implement agile processes and deliver faster and better to the customer.

There are multiple definitions available on Internet for DevOps and neither they are wrong or incorrect. DevOps does not provide a framework or methodology. It is a set of principles and practices that when employed within an organization, engagement or project achieves the goal and vision of both DevOps and organization. These principles and practices do not mandate any specific process, tools and technologies and environment. DevOps provides the guidance which can be implemented through any tool, technology and process although

some of the technology and processes might be more applicable to achieve the vision of DevOps Principles and practices.

Although, DevOps practices can be implemented in any organization that provides services and products to customers, going forward in this book, we will look at DevOps from perspective of a Software development and operations department of any organization.

So, what is DevOps? DevOps is defined as.

- It is a set of principle and practices
- bringing both Developers and Operations team together from start of the software system
- for faster, quicker and efficient end-to-end delivery of software system
- to end customer again and again
- in a consistent and predictable manner
- reducing time to market thereby gaining competitive advantage

Read out loudly the above definition of DevOps and if you look at it closely, it does not indicate or refer to any specific processes, tools or technology. It is not prescribing any particular methodology or environment.

The goal of implementing DevOps principles and practices in any organization is to ensure that stakeholders (including customers) demand and expectation are met efficiently and effectively.

Customer's demand and expectations are met when

- Customer gets the features they want
- Customer get the feature when they want
- Customer get the faster updates on features
- The quality of delivery is high

When an organization can meet above expectations, customers are happy and remains loyal to the organization. This in turn increases the market competitiveness of the organization which results in bigger brand and market valuation. It has a direct impact on top and bottom line of the organization. The organization can invest further on innovation and customer feedback, bring about continuous changes to its system and services to stay relevant.

The implementation of DevOps principles and practices in any organization is guided by its surrounding ecosystem. This ecosystem is made of the industry and domain the organization belong to.

DevOps is based on a set of principles and practices. We will look into details about these principles and practices later in this chapter. The core principles of DevOps are

- Agility
- Automation
- Collaboration
- Feedback

And core DevOps practices are

- Continuous Integration
- Configuration management
- Continuous Deployment
- Continuous Delivery
- Continuous learning

DevOps is not a new paradigm however, it is gaining lot of popularity and traction in recent times. Its adoption is at its highest level and more and more companies are undertaking this journey. I purposely mentioned DevOps as a journey because there are different levels of maturity within DevOps. While successfully implementing Continuous Deployment and Delivery are considered as highest level of maturity in this journey adopting source code control, agile software development is considered as a beginning.

One of the first thing DevOps talks about is *breaking the barriers between Dev and Operations team*. It brings about the close collaboration aspect between multiple teams. It is about breaking the mindset that Dev is responsible for writing code only and pass it on to operations for deployment once it is tested. It is also about breaking the mindset that Operations have no role to play in development activities. Operations should influence the planning of the product and should be aware of the features coming up as release. They should also continually provide feedback to Dev on the operational issues such that they can be fixed in subsequent releases They should influence the design of system for better operational working of the system. Similarly, Dev should help the operations in deployment of the system and also solve incidents when they arise.

The definition talks about *faster, quicker and efficient end to end delivery of systems to stakeholders*. It does not talk about how fast, quick or efficient the delivery should be. It should be fast or quick enough depending on the organization domain, industry, customer segmentation and more. For some organization fast enough could be quarterly while for others it could be weekly. Both types are valid for DevOps point of view and they can deploy relevant processes and technology to achieve the same. DevOps does not mandate it. Organizations should identify the best implementation of DevOps principles and

practices based on their overall project, engagement and organization vision.

The definition also talks about *end to end delivery*. It means that from the planning and delivery of the system to the services and operations should be part of DevOps implementation. The processes should be such that it allows for greater flexibility, modularity and agility in application development lifecycle. While organizations are free to use the best fit process – Waterfall, agile, Kanban and more, typically organization tends to favor agile process with iterations based delivery. This allows for faster delivery in smaller units which are far more testable and manageable compared to large big delivery.

DevOps talks about *to end customer again and again in a consistent and predictable manner*. This means that organization should continually deliver to customer with newer and upgraded features using automation. We cannot achieve consistency and predictability without the use of automation. Manual work should be reducing to none to ensure high level of consistency and predictability. The automation should also be end to end to avoid failures. This also indicates that the system design should be modular allowing faster delivery while they are reliable, available and scalable. Testing plays a great role in consistent and predictable delivery.

The end result of implementing the before mentioned practices and principles is that organization is able to meet the expectations and demand of customers. Organization is able to grow faster than competition and further increase the quality and capability of their product and services through continuous innovation and improvement.

DevOps Principles

DevOps is based on a set of foundational beliefs, culture and processes. These forms the pillars on which it is built upon. These provide a natural ecosystem to bring delivery excellence within an organisation. Let's look briefly into some of the these important principles

Collaboration and communication

One of the prime tenets of DevOps is collaboration. Collaboration means that different team comes together to achieve the common objective. It defines clear roles and responsibilities, overall ownership, accountability and responsibility for the team. The team comprises of both Development and operational people. Both together are responsible for delivering rapid high quality release to the end stakeholders.

Both the teams are part of end to end application lifecycle process. The Operations team are

part of the planning exercise for the features, providing their feedback on overall operational readiness and issues of the business application and services. The development team must play role in operational activities. They must assist in deploying the release to production and provide support in terms of fixing production issues. This kind of environment and ecosystem fasters continuous feedback and innovation. There is a shared vision and everybody in the team in achieving the same.

Agility towards change

Agility refers to flexibility and adaptability aspects for people, process and technology. People should have open mindset to accept change, play different roles, take ownership and accountability. Process would generally refer to

- Application lifecycle management
- development methodology and
- software design.

Application Lifecycle Management

Wikipedia defines application lifecycle management as "Application lifecycle management (ALM) is the product lifecycle management (governance, development, and maintenance) of computer programs. It encompasses requirements management, software architecture, computer programming, software testing, software maintenance, change management, continuous integration, project management, and release management."

Application Lifecycle Management refers to the Management of planning, gathering requirements, building and hosting code, testing code in terms of code coverage, unit tests, versioning of code, releasing code to multiple environments, tracking and reporting, functional tests, environment provisioning, deployment to production and operations for business applications and services. The operational aspects would include monitoring, reporting and feedback activities. Overall, ALM is a huge area and comprises of multiple activities, tools and processes. Special attention should be provided to craft appropriate application lifecycle steps to induce confidence on the final deployed system. For example, processes can be implemented which mandates that code cannot be checked in source code repository if unit tests do not pass completely. ALM comprises of multiple stages like Planning, Development, testing, deployment and operations.

In short, ALM defines a process to manage an application from its conception to delivery and integrates multiple teams together to achieve a common objective. A typical Application lifecycle management is shown in figure 1. ALM is a continuous process that

starts with planning of iterations, building and testing the iteration, deploy it on production environment and provide after deployment services to the customer. The feedback from customers and operations are passed on to the planning them which eventually incorporates them into subsequent iterations and this process loop continues.

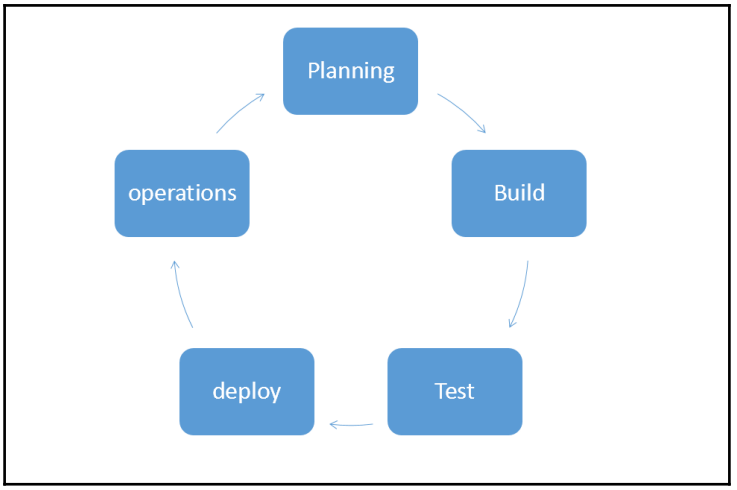

Figure 1: Application Lifecycle Management phases

Development methodology

Development methodology should be flexible and elastic to enable multiple smaller iterations or sprints of delivery. Each sprint and iteration must be functionally tested. Smaller iterations help in completing specific smaller features and pushing them the production. This provides team clear sense of direction, clear scope of work, setting up of expectations along with ownership of the release.

Software design

Software Design should implement architectural principles that fosters modularity, decomposition of large functionality into smaller features, reliability, high availability, scalability, audit capabilities, monitoring to name a few.

Process and tools automation

Automation plays a great role in achieving the overall DevOps goals. Without automation, DevOps cannot achieve its end objectives. Automation should be implemented for the entire Application lifecycle management starting for building the application to the delivery and deployment to the production environment. Automation brings cadence and high level of confidence that the output for each step in application lifecycle management if of high quality, robust and it is relatively risk free to deploy on production. Automation also help sin rapid delivery of business application to multiple environments because it is capable of running multiple build processes, execute thousands of unit tests, figure out code coverage comprising of millions of lines of code, provision environments, deploy applications and configure them at desired level.

Fail fast and early

At first glance, sounds weird talking about failing in a DevOps book that is supposed to make delivery of software successful. Trust me, it is not! Failing fast and early refers to the process of finding issues and risks as early as possible within the Application lifecycle development. Not knowing the issues that arises towards the end of ALM cycle is an expensive affair because a lot of work would have happened on it. It might involve design and architectural changes which can jeopardize the viability of entire release. If the issues can be found at the beginning phase, they can be resolved without much impact to the release. Automation plays a big role in identifying the issues early and fast.

Innovation and continuous learning

DevOps fosters the culture of innovation and continuous learning. There is constant feedback flow regarding the good and bad, what's working and what's not working on various environments. The feedback is used to try out different things either to fix the existing issues or find better alternatives. Through this exercise, there is constant information flow about how to make things better and that in turn provides impetus to find alternate solutions. Eventually, there are break-through findings and innovation that can further be developed and brought to production.

DevOps practices

DevOps consists of multiple practices each providing distinct functionality to the overall process. Figure 2 shows the relationship between them. Configuration management,

Continuous integration and Continuous Deployment form the core practices that enables DevOps. When we deliver software services combining these three services, we achieve continuous delivery. Continuous delivery is a capability and maturity of an organization depended on the maturity of Configuration management, Continuous integration and Continuous deployment. Continuous feedback at all stages forms the feedback loop that helps in providing superior services to the customers. It runs across all DevOps practices. Let's deep dive into each of these capabilities and DevOps practices.

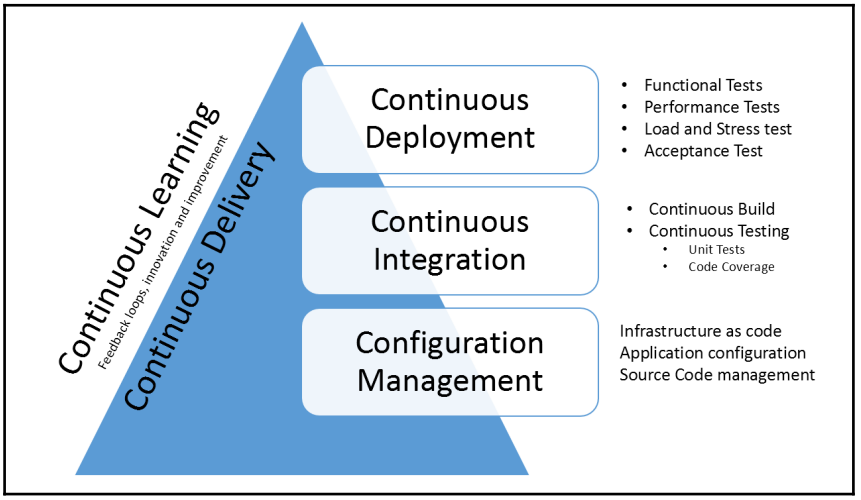

Figure 2: DevOps practices and their activities

Configuration management

Business application and services needs an environment on which they can be deployed. Typically, the environment is an infrastructure comprising on multiple server, compute, network, storage, containers and more working together such that business application can be deployed on top of them. Business applications are decomposed into multiple services running on multiple servers either on-premise or on the clouds and each service has its own configuration along with requirements related to infrastructure configuration. In short, both infrastructure and application is needed to deliver systems to customers and both of them have their own configuration. If the configuration drifts, the application might not work as expected leading to downtime and failure. Moreover, as ALM process dictates use of multiple stages and environment, an application would be deployed to multiple environment with different configurations. The application will be deployed to

development environment for developers to see the result of their work. The application will be deployed to multiple test environment with different configuration for functional tests, load and stress tests, performance tests, integration tests and more, it would also be deployed to pre-production environment to conduct user acceptance tests and finally on production environment. It is important that application can be deployed to multiple environments without undertaking any manual changes to its configuration.

Configuration management provides a set of processes and tools and they help in ensuring that each environment and application gets its own configuration. Configuration management tracks configuration items and anything that changes from environment to environment should be treated as Configuration item. Configuration management also defines the relationships between the configuration items and how changes in one configuration item will impact other configuration item.

Configuration management helps in following

- **Infrastructure as Code**: When the process of provisioning of infrastructure and its configuration is represented through code and the same code goes through application lifecycle process, it is known as Infrastructure as code. Infrastructure as code helps in automating the provisioning and configuration of infrastructure. It also represents the entire infrastructure in code that can be stored in a repository and version controlled. This allows to use previous environment configurations when needed. It also enables provisioning of environment multiple times in a consistent and predictable manner. All environments provisioned through this way are consistent and equal in all ALM stages.
- **Deployment and configuration of application:** Deployment of application and its configuration is the next step after provisioning of infrastructure. Examples of application deployment and configuration is to deploy a webdeploy package on a server, deploy sql server schemas and data (bacpac) on another server, change SQL connection string on web server to represent appropriate SQL Server. Configuration Management stores values for application configuration for each environment on which it is deployed.

The configuration applied should also be monitored. The expected and desired configuration should be consistently maintained. Any drift from this expected and desired configuration would render the application as not available. Configuration management are also capable of finding the drift and re-configure the application and environment to its desired state.

With automated Configuration Management in place, nobody in the team has to deploy and configure the environments and applications on production. Operations team is not reliant on development team or long deployment documentation,

Another aspect of configuration management is Source code control. Business application and services comprises of code and other artifacts. Multiple team members work on the same and same files. The source code should be up to date at any point of time and should be accessible by only authenticated team members. The code and other artifacts by themselves configuration items. Source control helps in collaborate and combination within the team since everybody is aware of what other person is doing and conflicts are resolved at an early stage.

Continuous integration

Multiple developers write code that is eventually stored in a common repository. The code is normally checked in or pushed to repository when developer has finished developing his feature. This can happen in a day or might take days or weeks. Some of the developers might be working on the same feature and they might also follow the same practices of pushing/checking-in code in days or weeks. This can cause issues with the quality of code. One of the tenet of DevOps is to fail fast. Developers should check-in/push their code to repository often and compile the code to check if he/she has not introduced any bug and that the code is compatible with code written by his/her fellow member. If developer do not follow this practice, then the code on their machine will grow very large difficult to integrate will other's code. Moreover, if the compile fails, it is difficult and time consuming to fix the issues arising out of it.

Continuous integration solves these kind of challenges. Continuous integration helps in compilation and validation of the code pushed/check-in by a developer by taking it through a series of validation steps. Continuous integration creates a process flow consisting of multiple steps. Continuous integration is comprised of continuous automated build and continuous automated tests. Normally the first step is compilation of the code. After successful compilation, each step is responsible for validating the code from a specific perspective. For example, unit tests can be executed on the compiled code, code coverage can be executed to check which code paths are executed by unit tests. These could reveal if comprehensive unit tests are written or there is scope to add further unit tests. The end result of continuous integration are deployment packages that can be used by continuous deployment for deploying them to multiple environments.

Developers are encouraged to check-in their code multiple times in a day instead of days or weeks. Continuous integration would initiate the execution of the entire pipeline immediately as soon as the code is checked-in or pushed. If compilation succeeds, code tests and other activities that are part of the pipeline are executed without error, the code is deployed to a test environment and integration tests are executed on it. Although every system demands its own configuration of Continuous Integration, a minimal sample Continuous integration is shown in Figure 3.

Continuous integration increases the productivity of the developers. They do not have to manually compile their code, run multiple types of tests one after another and then create packages out of it. It also reduces the risk of getting bugs introduced in code and code does not get stale. It provides early feedback to the developers about the quality of their code. Overall, the quality of deliverables is high and deliverables are delivered faster by adopting continuous integration practice.

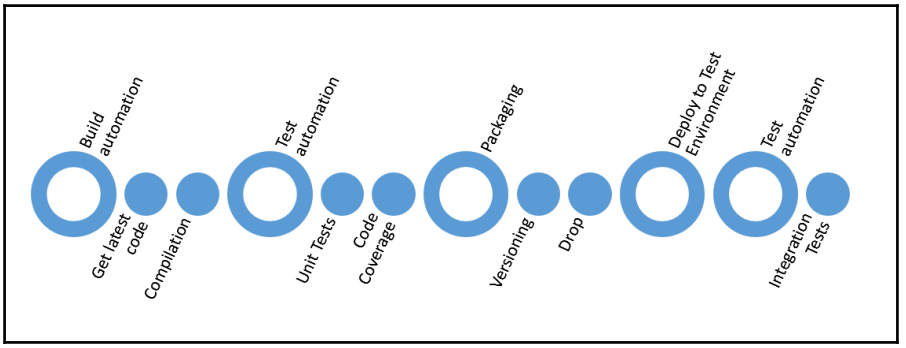

Figure 3: Sample Continuous Integration process

Build automation

Build automation consists of multiple tasks executing in sequence. Generally, the first task is responsible for fetching latest source code from repository. The source code might comprise of multiple projects and files. They are compiled to generate artifacts like executable, dynamic link libraries, assemblies and more. Successful build automation reflects that there are no compile time errors in code.

There could be more steps in Build automation depending on nature and type of project.

Test automation

Test automation consists of tasks that are responsible for validating different aspects of code. These tasks are related to testing code from different perspective and are executed in sequence. Generally, the first step is to run a series of unit tests on the code. Unit testing refers to process of testing the smallest denomination of a feature validating its behaviour in isolation from other features. It can be automated or manual however the preference is towards automated unit testing.

Code Coverage is another type of automated testing that can be executed on code to find out how much of code is executed while running the unit tests. It is generally represented as a percentage and refers to how much of code is testable through unit testing. If code coverage is not close to hundred percent, it is either because the developer has not written unit tests for that behaviour or the uncovered code is not required at all.

Successful execution of Test automation resulting in no significant code failure should start executing the Packaging tasks. There could be more steps in Test automation depending on nature and type of project.

Application packaging

Packaging refers to the process of generating deployable artifacts like msi, NuGet and web deploy packages, database packages, versioning them and storing them at location such that they can be consumed by other pipelines and processes.

Continuous deployment

By the time, the process reaches Continuous Deployment, Continuous integration has ensured that we have fully working bits of an application that can now be taken through different Continuous Deployment activities. Continuous Deployment refers to the capability of deploying business applications and services to pre-production and production environments through automation. For example, continuous deployment could provision and configure the pre-production environment, deploy application to it and configure. After conducting multiple validation like functional tests, performance tests on pre-production environment, production environment is provisioned, configured and the application is deployed to Production environments through automation. There are no manual steps in deployment process. Every deployment task is automated. Continuous deployment can provision the environment and deploy the application for a bare metal deployment while it could reuse existing environment and conduct only application deployment if the environment already exists. It is always better to conduct bare metal green field deployment however, business justification can demand for brown field deployments.

All the environments are provisioned through automation using Infrastructure as code. This ensure that all environments whether its Dev, test, pre-production, production an any other environment are same. Similarly, the application is deployed through automation ensuring that it is also deployed uniformly across all environments. The configuration across these environments could be different for the application.

Continuous Deployment is generally integrated with Continuous integration. When

continuous integration has done its work by generating the final deployable packages, continuous deployment kicks in and start its own pipeline. This pipeline is called the *Release pipeline*. Release pipeline consists of multiple environments with each environment consisting of tasks responsible for provision of environment, configuration of environment, deploying applications, configuring applications, executing operational validation on environments and test the application on multiple environments. We will look in Release pipeline in greater details in next chapter and also chapter on Continuous deployment.

Employing Continuous deployment provides immense benefits. There is high level of confidence in the overall deployment process which helps in faster and risk free releases on production. The chances of anything going wrong comes down drastically. The team would have lower stress levels and rollback to previous working environment is possible if there are issues in current release.

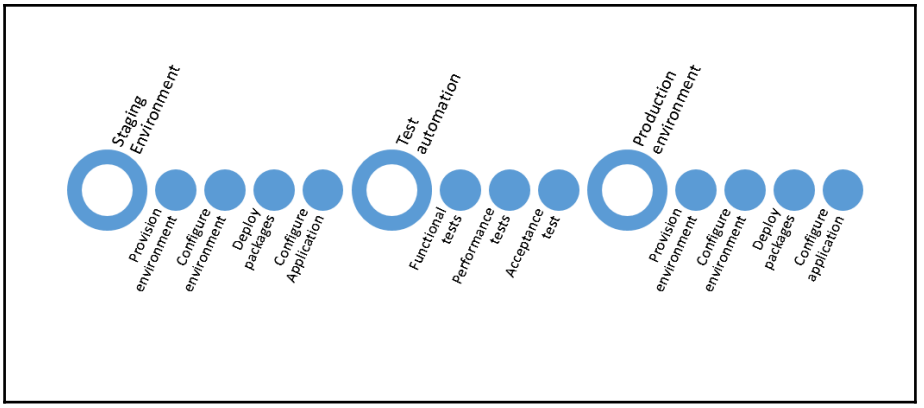

Figure 4: Sample continuous Deployment/ Release pipeline process

Although every system demands its own configuration of Release pipeline, a minimal sample Continuous Deployment is shown in Figure 4. It is important to note that generally provisioning and configuring of multiple environment is part of release pipeline and approvals should be sought before moving to next environment. The approval process might be manual or automated depending on the maturity of the organisation.

Test environment deployment

The Release pipeline start once the drop is available from continuous integration and the first step it should take is to get all the artifacts from the drop. After which, it might create a completely new bare metal test environment or reuse an existing one. This is again

dependent on type of project and nature of testing planned to be executed on this environment. The environment is provisioning and configured. The application artifacts are deployed and configured.

Test automation

After deploying application, a series of tests can be performed on the environment. One of the tests executed here is Functional tests. Functional tests are primarily aimed for validating the feature completeness and functionality of the application. These tests are written from requirements gathered from customer. Another set of tests that can be executed are related to scalability and availability of the application. This typically includes load test, stress tests and performance tests. It should also include operational validation of the infrastructure environment.

Staging environment deployment

This is very similar to the Test environment deployment with the only difference is that the configuration values for the environment and application would be different.

Acceptance tests

Acceptance tests are generally conducted by stakeholders of the application and this can be manual or automated. This step is a validation from customer point of view about the correctness and completeness of application functionality.

Deployment to production

Once the customer provides its approval, same steps as that of Test and staging environment deployment are executed, with the only difference that the configuration values for the environment and application are specific to production environment. A validation is conducted after deployment to ensure that application is running according to expectation.

Continuous delivery

Continuous Delivery and Continuous Deployment might sound similar to many readers; however, they are not the same. While Continuous Deployment talks about deployment to multiple environments and finally to production environment through automation,

Continuous Delivery practices is the ability to generate application packages in a way that are readily deployable in any environment. For generating artifacts that are readily deployable, continuous integration should be used to generate the application artifacts, a new or existing environment should be used for deploying these artifacts, conduct functional tests, performance tests and user acceptance test through automation. Once these activities are successfully executed with no errors, the application package is referred as readily deployable. Continuous delivery comprises of continuous integration along with deployment to an environment for final validations. It helps in getting feedback faster from both the operations as well as from end user. This feedback can then be used to implement in subsequent iterations.

Continuous learning

With all the before mentioned DevOps practices, it is possible to create great business application and deploy them automatically to production environment however, benefits of DevOps will not last for long, if continuous improvement and feedback principle is not in place. It is utmost important that real time feedback about the application behavior is passed on as feedback to the development team from both end users and operations team

Feedback should be passed to the teams providing relevant information about what is going well and importantly what is not going well.

Applications should be built with monitoring, auditing and telemetry in mind. The architecture and design should support these. The operations team should collect the telemetry information from the production environment, capture any bugs and issues and pass it on the Development team such that they can get fixed in subsequent releases. The same has been shown in Figure 5.

Continuous learning helps in making the application robust and resilient to failures. It helps in making sure that the application is meeting the requirements of the consumers.

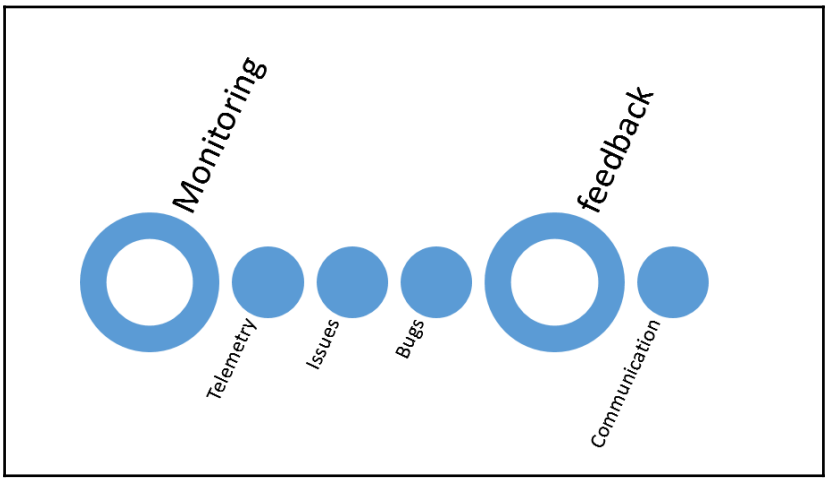

Figure 5: Sample Continuous Learning process

Measuring DevOps

Once DevOps practices and principles are implemented, the next step is to find out whether DevOps is providing any tangible benefits to the organization. To find the impact of DevOps on delivering changes to the customers few metrics needs to be tracked. Also, there should be some baseline data available for these metrics prior to DevOps implementation. After implementing DevOps, the same metrics should be captured over a period of time and then compared with the baseline. This comparison of data should bring out intelligence whether DevOps is effective in the organization or there should be changes within the implemented DevOps practices.

Some of the important metrics that should be tracked are

Summary

In this chapter, we saw the problems plaguing the software organizations, definition of DevOps, how DevOps helps in eliminating these pain areas. We also went through the principles and practices of DevOps explaining briefly their purpose and usefulness. This chapter is also foundation and backbone for rest all the chapters. All chapters henceforth will be step by step realization of the principles and tenets we talked about in this chapter.

Although this chapter was heavy on theory subsequent chapters will start delving in technology and practical steps to implement DevOps. You should now have a good grasp of Devops concepts.

From next chapter onwards, we will get into details of tools and technologies that enables DevOps.

2
DevOps Tools and Technologies

In the last chapter, we saw the problems faced by organizations in delivering software services to customers. We understood the meaning of DevOps and how it addresses the challenges of software delivery. We went through understanding foundational principles on which DevOps is based on and we discussed the practices and pillars through which DevOps achieves its end goal.

This book is about practical implementation of DevOps through technology and realize its benefits. Technology is an enabler for DevOps. Technology helps in DevOps in following ways

- Enables faster collaboration and communication among teams- makes them more efficient and effective.
- Helps in faster, better and automated process implementation.
- Consistent and predictable automated delivery. Brings higher cadence and confidence in the delivery process.
- Feedback backed up by telemetry
- Agile deployments.

This chapter and next two chapters will introduce foundational platform and technologies instrumental in enabling and implementing DevOps practices. These include

- Technology stack for implementing Continuous integration, Continuous deployment, Continuous delivery and Configuration management and continuous improvement. These form the backbone for DevOps processes and includes source code service, build services, release services through Visual Studio Team Services.

- Platform and technology used to create and deploy a sample web application. It includes technologies like Microsoft .NET, ASP.NET and Sql Server Databases.
- Tools and technology for configuration management, testing code and application, authoring infrastructure as code and deployment of environments. Examples of these tools and technologies are Pester for environment validation, environment provisioning through Azure Resource Manager (ARM) templates, Desired State Configuration (DSC) and Powershell, application hosting on containers through Windows Containers and Docker, Application and database deployment through Webdeploy packages and Sql Server bacpac' s.

Cloud technology

Cloud is ubiquitous. It is used throughout in this book. Cloud is used for our development environment, implementation of DevOps practices and deployment of applications.

Cloud is relatively a new paradigm in infrastructure provisioning, application deployment and hosting space. The only options prior to advent of cloud was either self-hosted on-premise deployments or using services from a hosting service provider. However, cloud is changing the way enterprises look at their strategy related to infrastructure and application development, deployment and hosting. In fact, the change is so enormous that it has found its way into every aspect of an organization's software development processes, tools and practices.

Cloud computing refers to practice of deploying applications and services on internet with a cloud provider. A cloud provider provides multiple types of services on cloud. They are divided into three categories based on their level of abstraction and degree of control on services. These categories are

- Infrastructure as a Service (also popularly known as IaaS).
- Platform as a Service (also popularly known as PaaS).
- Software as a Service (also popularly known as SaaS).

These three categories differ based on the level of control a cloud provider exercises compared to cloud consumer. The services provided by a cloud provider can be divided into layers with each layer providing a type of service. As we move higher in stack of layers, the level of abstraction increases and increases cloud provider's control on services. In other words, the cloud consumer start losing control on services as you move higher in each column.

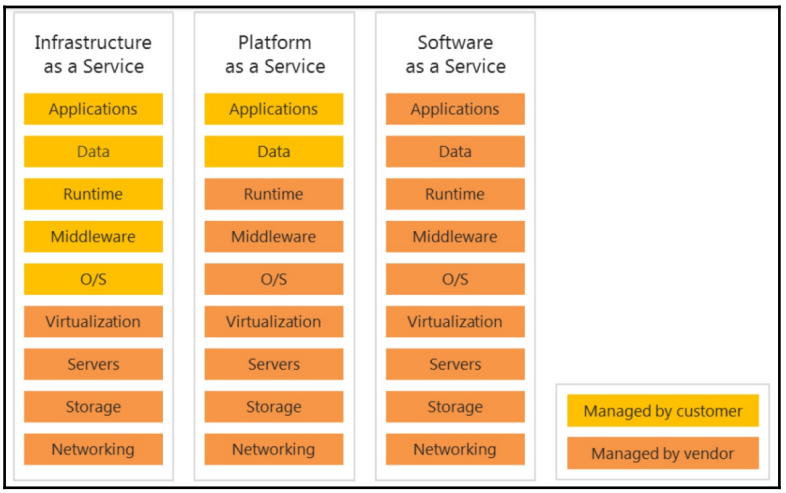

Figure 1: Cloud Services – IaaS, PaaS and SaaS

Figure 1 shows the three types of service available through cloud providers. It shows the layers that comprises these services. These services are stacked vertically to each other and shows the level of control a cloud provider has compared to a consumer. From Figure 1, it is clear that for IaaS, a cloud provider is responsible for providing, controlling and managing layers from hardware layer up to virtualization layer. Similarly, for PaaS, a cloud provider controls and manages from hardware layer up to the runtime layer while the consumer controls only the application and data layer.

Infrastructure as a Service(IaaS)

As the name suggests, Infrastructure as a Service are infrastructure service provided by a cloud provider. These services include the physical hardware and its configuration, network hardware and its configuration, storage hardware and its configuration, load balancers, compute and virtualization. Any service above virtualization is responsibility of consumer to provision, configure and manage. The consumer can decide to use the provided underlying infrastructure the way best suited to them. Consumers can consume the storage, network and virtualization to provision their virtual machines on top of them. It is consumer's responsibility to manage and control the virtual machines and things deployed within it.

Platform as a Service(PaaS)

Platform as a service enables consumers to deploy their applications and services on the provided platform, consuming the underlying runtime, middleware and services. The cloud provider provides the services from infrastructure till runtime services. The consumers cannot provision virtual machines as they do not access and control over them instead they can only control and manage their applications. This is comparatively faster way of development and deployment because now the consumer for focus on application development and deployment. Examples of Platform as a Service includes Azure automation, Azure Sql, Azure App Services.

Software as a Service(SaaS)

Software as a service provides complete control on the service to the cloud provider. The cloud provider provisions, configures and manages everything from infrastructure to the application. It includes provisioning of infrastructure, deployment and configuration of application and provides application access to the consumer. The consumer does not control and manage the application. The consumers can use and configure only parts of the application. They control only their data and configuration. Generally, multi-tenet applications used by multiple consumers like Office 365 and Visual Studio team services are examples of SaaS.

Advantages of using Cloud computing

There are multiple distinct advantages of using Cloud technologies. The major among them are.

1. **Cost Effective**: Cloud computing helps in organizations to reduce their cost of storage, networks and physical infrastructure. They do not have to buy expensive software licenses as well. The operational cost of managing these infrastructures also reduces due to lessor effort and manpower requirements.
2. **Unlimited capacity:** Cloud provides unlimited resources to its cloud consumer. This makes sure that application will never get throttled due to limited resource availability.
3. **Elasticity:** Cloud computing provides the notion of unlimited capacity and applications deployed on it can scale-up and down on need basis. When demand for application increases, cloud can be configured to scale up the infrastructure and application by adding additional resources. At the same time, it can scale down resources not needed during low demand.

4. **Pay as you go:** Using cloud eliminates capital expenditure and organizations pay only for what they use thereby providing maximum return on investment. Organizations do not need to build additional infrastructure to host their application for times of peak demand.
5. **Faster and better:** Cloud provides ready to use applications and faster provisioning and deployments of environments. Moreover, organizations get better managed services from cloud provider with higher Service level agreements.

We will use *Azure* as our Cloud computing provider and platform as our preferred provider for the purpose of demonstrating samples and examples. However, you can use any cloud provider that provider complete end to end services for DevOps.

We will use multiple features and services provided by Azure across IaaS and PaaS. We will consume Operational Insights and Application Insights for monitoring of our environment and application. They will help in capturing relevant telemetry for auditing purpose. We will provision Azure virtual machines running Windows and Docker containers as hosting platform. Windows server 2016 as target operating system for our applications on cloud. Azure Resource Manager (ARM) (we will look into it in details in next chapter) as our Cloud technology framework and use Infrastructure as code in form of ARM templates for our deployments. We will also use Desired state configuration and Powershell as our configuration management platform and tool.

We will use Visual studio team services (VSTS), a suite of PaaS services on cloud provided by Microsoft to set up and implement our end to end DevOps practices. Microsoft also provides same services as part of Team Foundation Services (TFS) as an on-premise solution.

Technologies like Pester, DSC and Powershell can be deployed and configured to run on any platform. They will help in validation of environments and in configuration both application and environment as part of Configuration Management.

We will create a sample application in chapter 5 and the entire application life cycle management will be implemented through DevOps practices. Windows Server 2016 is our target platform for deploying the application. Windows Server 2016 is a breakthrough operating system from Microsoft also referred as Cloud Operating system. We will look into Windows Server 2016 next.

Windows Server 2016

Windows Server 2016 has come a long way. All the way from Windows NT to Windows 2000 and 2003, then windows 2008 (R2) and 2012 (R2) and now Windows Server 2016. WinNT was the first popular windows server among the enterprises however the true enterprise server was Windows 2000 and Windows 2003. The popularity of windows server 2003 was unprecedented and it was adopted widespread. With Windows Server 2008 and 2008 R2, the idea of datacenter took priority and enterprises with their own datacenter adopted it. Even Windows Server 2008 series were quite popular among the enterprises. In 2010, Microsoft cloud Azure was launched. The first step towards cloud operating system was windows Server 2012 and 2012 R2. They had the blueprints and technology to be provisioned on Azure seamlessly. Now, when Azure and cloud adoption is gaining enormous popularity, Windows Server 2016 is released as a true Cloud operating system. The evolution of Windows Server is shown in Figure 2.

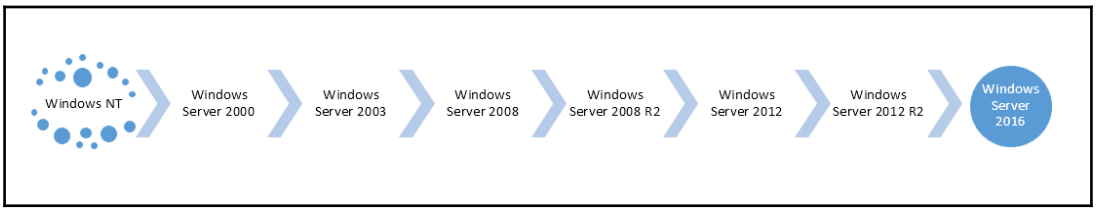

Figure 2: Windows Server evolution

Windows Server 2016 is referred as Cloud operating system. It is built with cloud in perspective. It is also referred as the first operating system that enables DevOps seamlessly by providing relevant tools and technologies. It makes implementing DevOps simpler and easier through its productivity tools. We will next look briefly into these tools and technologies.

Multiple choices for Application platform

Windows Server 2016 comes with many choices for application platform for applications It provides:

- Windows Server 2016 Server
- Nano Servers
- Windows and Docker containers

- Hyper-V containers
- Nested virtual machines

Windows server as hosting platform

Windows server 2016 can be used in way it has always been used i.e. hosting applications and providing server functionalities. It provides the necessary service to make applications secure, scalable, highly available, virtualization, directory services, certificate services, Web Server, databases and more. These services can be consumed by the enterprise services and application.

Nano servers

Windows Server provides a new option to host application and services. This is a new variety of lightweight scaled down windows server containing only the needed kernel and drivers to run them as operating system. They are also known as head-less servers. They do not have any graphical user interface and the only way to interact and manage them it through remote Powershell. Out of the box, they do not contain any service or feature. The services need to be added to Nano servers explicitly before using them. They are so far the most secure servers from Microsoft. They are very light weight and the resource requirements and consumption is less than 80% of a normal windows server. The number of services running, the number of ports open, the number of processes running and the amount of memory and storage required are less than 80% compared to normal Windows server.

Even though Nano Servers out of box just has the kernel and drivers, they can be enhanced in capabilities by adding features to it and deploy any windows application on it.

Windows Containers and Docker

Containers are one of the most revolutionary feature added to Windows Server 2016 after Nano Servers. With the popularity and adoption of Docker containers which primarily runs on Linux, Microsoft decided to came up with container services on Windows Server 2016.

Containers are operating system virtualization. It means that multiple containers can be deployed on same operating system and each one of them will share the host operating system kernel. It is the next level of virtualization after server virtualization (virtual machines). Containers generate the notion of complete operating system isolation and independence although it uses the same host operating system underneath it. This is possible through the use of namespace isolation and image layering. Containers are created

from images. Images are immutable and cannot be modified. Each image has a base operating system and then a series of instructions are executed against it. Each instruction creates a new image on top of previous image and contains only the modification. Finally, a writable image is stacked on top of these images. These images are combined into a single image which can then be used for provisioning of containers. It is shown in Figure 3.

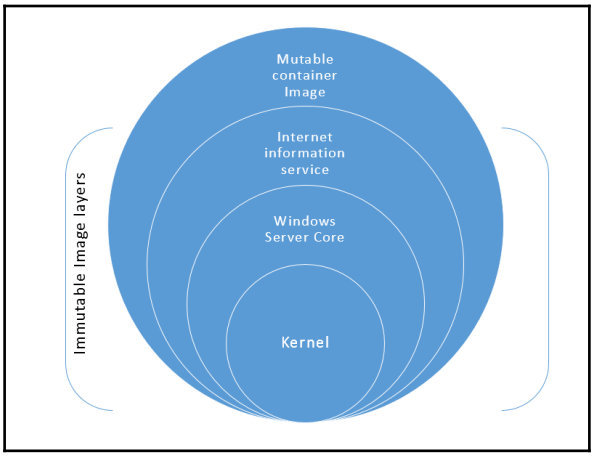

Figure 3: Containers made up of multiple image layers

Namespace isolation helps in providing containers pristine new environments. Neither the containers have can see the host resources not the host can view the container resources. For the application within the container, a complete new installation of the operating system is available. The containers share the host's memory, CPU and storage.

Containers offer operating system virtualization which means the containers can host only those operating system supported by the host operating system. There cannot be a windows container running on a Linux host while a Linux container cannot run on a windows host operating system.

Docker provides container services on Windows Server 2016. It provides both the API's and the client to manage the containers. It helps in building the images and creating containers from images. It helps in starting, stopping and removing containers. Containers are super-fast from deployment perspective as they do not require booting of operating system, it reuses the running host operating system. It means containers demand less maintenance compared to virtual machines. We will learn more details on container and Docker in next chapter.

Hyper-V Containers

Another type of container technology windows server 2016 provides is Hyper-V containers. These containers are similar to Windows containers. They are managed through the same Docker client and extends the same Docker API's however these containers contain their own scaled down operating system kernel. They do not share the host operating system but has their own dedicated operating system and have their own dedicated memory and CPU assigned exactly the way virtual machines are assigned resources.

Hyper-V containers brings in higher level of isolation of containers from the host. While Windows containers runs in full trust on the host operating system, Hyper-V containers does not have full trust from the host perspective. It is this isolation that differentiates Hyper-V container with windows containers.

Hyper-V containers are ideal for hosting applications that might harm the host server effecting every other containers and services on it. Scenario's where users can bring in their own code and execute are examples of such applications. Hyper-V containers provides adequate isolation and security to ensure that applications cannot access the host resources and change them.

Nested virtual machines

Another breakthrough innovation of Windows Server 2016 is that now virtual machines can host virtual machines. Now, we can deploy multiple virtual machines containing all tiers of an application within a single virtual machine. This is made possible through software defined networks and storage.

Enabling Microservices

Nano servers and containers helps in providing advance lightweight deployment options through which we can now decompose the entire application into multiple smaller independent services, each with their own scalability and high availability configuration and deploy them independent of each other. Microservices helps in making the entire DevOps life cycle agile. With Microservices, changes to services does not demand that every other Microservices undergo every test validation. Only the changed service needs to be tested rigorously along with its integration with other services. Compare this to a monolithic application. Even a single small change will result in testing the entire application. Microservices helps in creating smaller teams for its development, testing of a service can happen independent of other services and deployment can be done for each service in isolation.

Continuous integration, continuous deployment and delivery for each service can be executed in isolation rather than compiling, testing and deploying the whole application every time there is a change.

Reduced maintenance

Because of their inherent nature, Windows Nano servers and containers are lightweight fast to provision. It helps in faster provisioning and configuration of environments thereby reducing the overall time taken by continuous integration and deployment. Also, these resources can be provisioned on Azure on demand without waiting for long hours. Because of their small footprint in terms of size, storage, memory and features, they need lesser maintenance. These servers are patched less often, patched with lesser fixes, they are secure by default, have lesser chances of failing applications makes them ideal for operations. The operations need to spend fewer hours maintaining these servers compared to normal servers. This reduces the overall cost for the organization and helps in DevOps will high quality delivery.

Configuration management tools

Windows Server 2016 comes with Windows Management Framework 5.0 installed by default. Desired state configuration(DSC) is the new configuration management platform is available out of box in Windows Server 2016. It has a rich mature set of features that enable configuration management for both environments and applications. With DSC, the desired state and configuration of environments are authored as part of Infrastructure as code and executed on a scheduled basis on every server. They help in checking the current state of servers with the documented desired state and helps in bringing them back to desired state. DSC is available as part of Powershell and Powershell helps in authoring these configuration documents.

Windows server 2016 provides Powershell unit testing framework known as PESTER. Historically, unit testing for infrastructure environments was always missing as a feature. Pester enables testing of Infrastructure provisioned – either manually or through Infrastructure as code using DSC configuration or ARM templates. They help in operational validation of the entire environment bringing in high level of cadence and confidence in continuous integration and deployment processes.

Deployment and packaging

Package Management and deploying utilities and tools through automation is a new concept in Windows world. Package management is ubiquitous in Linux world for a long time. Packing management helps in searching, saving, installing, deployment, upgrading, removal of software packages from multiple sources and repositories on demand. There are public repositories like Chocolatey, PSGallery available storing readily deployable packages. Tools like NuGet can connect these repositories and help in package management. They also help in versioning of the packages. Application relying on specific package version can download them on need basis. Package management helps in building the environments and application deployment. Package deployment is much easier and faster with this out of box windows feature.

Visual Studio Team Services

Now, it's time to focus on another revolutionary online service Visual studio team services (VSTS) that enables continuous integration, continuous deployment and continuous delivery seamlessly. In fact, it would be more appropriate to call it a suite of services available under a single name. VSTS is a PaaS provided by Microsoft and hosted on cloud. The same service is available as Team Foundation Services (TFS) on-premise. All examples used in this book uses Visual Studio team services.

According to Microsoft, VSTS is a cloud based collaboration platform that helps teams in sharing code, tracking work and ships software. VSTS is the new name and earlier it was known as visual studio online (VSO). VSTS is an enterprise software development tool and service that enables organizations in providing automation facilities to their end to end application life cycle management process from planning to deployment of application and getting real time feedback from software systems. This increases the maturity and capability of an organization to deliver high quality software systems to their customers again and again.

Successful software delivery involves efficiently bringing numerous processes and activities together. These includes executing and implementing various agile processes, increase collaboration among teams, seamless and automatic transition of artifacts from one phase of ALM to another phase, deployments to multiple environment. It is important to track and report on these activities to measure and take action and improve delivery process. VSTS makes it simple and easy. It provides a whole suite of services that enables

- Collaboration among every team member by providing a single interface for entire application life cycle management.
- Collaboration among development teams using source code management

services

- Collaboration among test teams using test management services
- Automation validation of code and packaging through continuous integration using Build management services
- Automating validation of application functionality, deployment and configuration of multiple environments through continuous deployment and delivery using Release management services
- Tracking and work item management using work management services.

Figure 4 shows all the services available from VSTS top navigation bar.

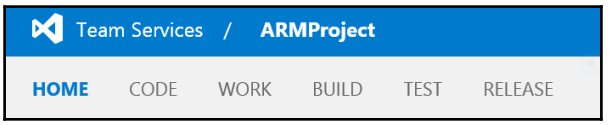

Figure 4: VSTS Services

Source code management service

Source code management services also known as Version control system is one of the flagship service from VSTS. Source code version control helps in storing the code in a repository that can either be centralized or distributed. It also helps in versioning of the code. Versioning helps in maintaining multiple copies of the same files with a new copy getting created when the code gets updated. It helps in viewing the history of the code, comparing code between different versions and help in retrieving previous versions.

We will need a VSTS account to be created before it can be used. We will look into details of creating a VSTS account in VSTS later in this chapter. Creation of an account creates a project collection in which every projects are created. The project collection is a container providing security boundary and additional services like agent pools. After an account is created, the next step is to create a project. After a project is created the browser is automatically redirected to project dashboard. Each project is based on a type of process. “Process” was earlier known as “Process Templates” in VSO. Process determines how the requirements are broken down into features or user stories and tasks. It also provides mechanism to manage them through work item tracking. There are three types of process available out of box in VSTS.

1. **SCRUM**: The SCRUM process is for teams who follows the SCRUM framework

for their application development life cycle.

2. **Agile:** The Agile process if for teams using Agile methods.
3. **CMMI:** The CMMI process is comparatively more formal process to executing projects. It majorly focuses on continuous improvement through telemetry.

Each of the process whether SCRUM, Agile and CMMI are different ways of executing projects and demands complete books by themselves. We will not go in details on them in this book.

The code link on top navigation bar will take us to the source code management control panel. This is shown in Figure 5.

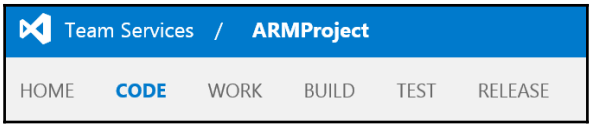

Figure 5: Source Code link

A project in VSTS is a security boundary and logical container that provides all the services we mentioned in previous section. VSTS allows for creation of multiple projects within a single account. By default, a repository is created with the creation of a project however, VSTS allows for creation of additional repositories within a single project. The relationship between VSTS account, project and repository is shown in Figure 6.

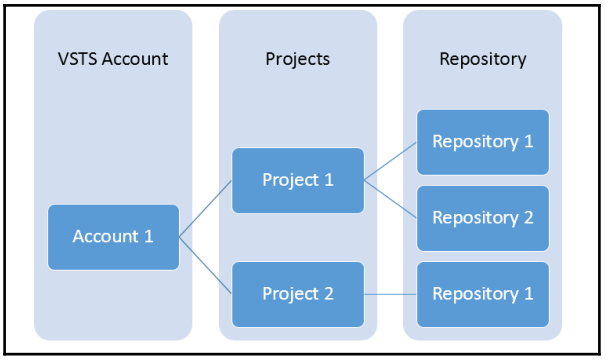

Figure 6: Relationship between VSTS Account, projects and repositories

VSTS provides two types of repository,

1. GIT
2. TFVC (Team foundation version control)

It also provides the flexibility to choose between GIT or TFVC source control repository. There can be combination of TFS and TFVC repositories available within a single project.

Team foundation version control(TFVC)

TFVC is the traditional and centralized way of implementing version control in which there is a central repository and developers work on it directly in connected mode to check-in their changes. If the central repository is offline or not available, developers cannot check-in their code and have to wait for it to online and available. Other developers can see only the checked in code. Developers can group multiple changes into a single changesets for checking-in code changes that are logical grouped to form a single change. TFVC locks the code files that are undergoing edits. Other developers can read the locked up file but they cannot edit it. They have to wait for the prior edit to complete and release the lock before they can edit. The history of check-ins and changes are maintained on the central repository while the developers have the working copy of the files but not the history.

TFVC works very well with large teams that are working on the same projects. This allow control over source code at a central location. It also works the best when the project is for long duration since the history is managed at a central location. TFVC has no issues working with large and binary files.

Exploring GIT

GIT on the other hand is modern distributed way of implementing version control where the developers can work on their own local copies of code and history in offline mode. Developers can work offline on their local clone of code. Each developer has a local copy of code and entire history and they work on their changes with this local repository. They can commit their code to the local repository. They can connect to the central repository for synchronization of their local repository on need basis. This allows every developer to work on any file since they would be working on their local copy. Branching in GIT does not create another copy of the original code and is extremely fast to create.

GIT works well with smaller team. With larger teams, there is a substantial overhead to manage multiple pull requests to merge the code on central repository. It also works best for smaller duration project as the history would not get large to be downloaded and manageable on every developer's local repository. Branching and merging is a breeze with

advance options.

GIT is the recommended way of using Source control because of the rich functionality it provides. We will use GIT as repository for our sample application in this book.

Build management service

Another very important service in VSTS is Build management services. The main task of Build services is to provide continuous integration services to projects.

As we already know by now, continuous integration is the process of building, compiling and validating code. It is about deploying code on to test environment and validate the code quality, code completion and whether the code bits are in working condition. Build services helps in automating this entire process. Similar to source code management, build services are scoped at project level. For each project, VSTS allows for creation of multiple build definitions and templates. Each build definition is attached to a particular branch of a repository. This could be a GIT or a VSTC repository. Templates provide the basic building blocks for definitions. The can be starting place to give a jump start for creating custom build definitions with activities already defined in the definition. A build definition can also be saved as a template and reuse to create other build definitions.

The build definition consists of multiple activities that run in sequence one after another like a pipeline. Each activity is responsible for executing a single task within the overall build pipeline. Figure 7 shows an example of build definition consisting of six tasks created using visual studio build template.

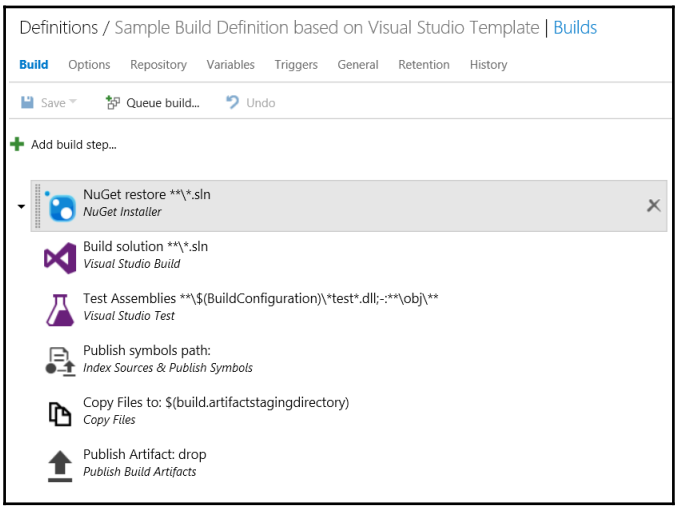

Figure 7: Sample build definition based on visual studio template

These tasks are responsible for

- Restoring the NuGet packages needed by the solution
- Build the entire solution which in turn builds every project within it.
- Execute unit tests on the compiled code.
- Publish symbol path that helps in debugging.
- Copy the generated and compiles assemblies to destination folder
- Publish the artifacts on VSTS server.

There are many more tasks available and can be used to augment the build definition further. Figure 8 shows the tasks available for build definition. There is also a market place from where more tasks can be made available to the build definitions. It is important to note at this stage is that same tasks are available both for Build as well as Release definitions. There is no different in their configuration and execution whether they are executed as part of build or release definition.

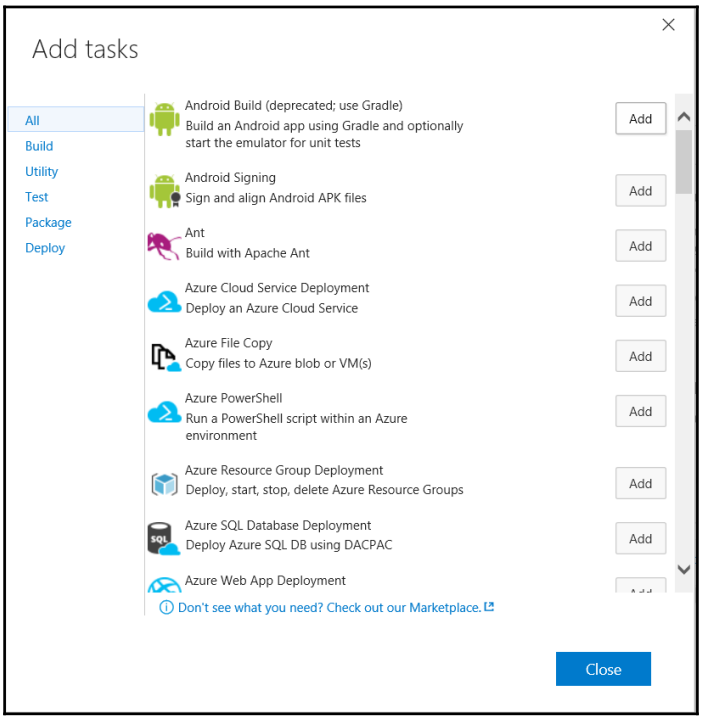

Figure 8: Build tasks

Executing Build Definitions

Build definitions needs build environment to be executed. The tasks in a build definition has pre-requisites and those should be available on the build servers. VSTS provides in-built build servers out of the box with pre-requisites installed on them. They are available to every account for executing build definitions. However, VSTS is flexible enough to accept our own provided custom build servers for execution of build definitions.

Build architecture

Figure 9 depicts the VSTS build architecture. Every build server should have build agents installed and running on them. Build agents are windows services configured to interact and work with VSTS build service. The agents are grouped together to for agent pools. An agent pool is a group of agents defined at VSTS account level and there can be multiple

agent pools defined. VSTS provides a default agent pool named “hosted”. Every VSTS provided servers are part of hosted agent pool.

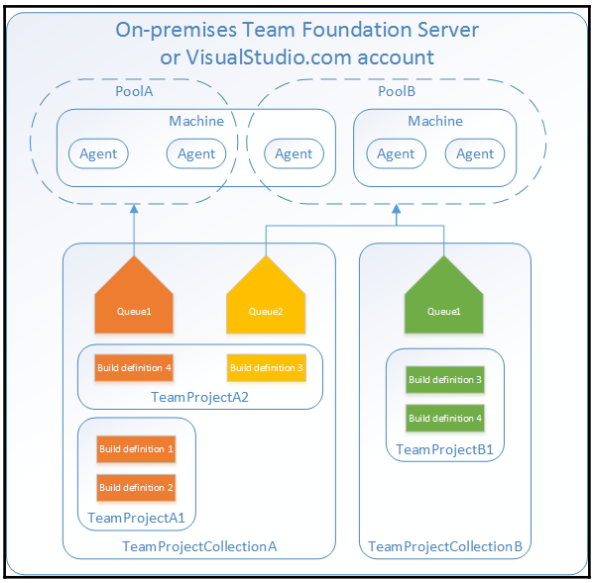

Figure 9: Relationship between Agents, Pools and Queue

Executing Build Definitions

Build definitions needs build environment to be executed. The tasks in a build definition demands capabilities and those should be available on the build environment. VSTS provides in build servers out of the box with pre-requisites available on them. They are available to every accounts in VSTS for executing build definitions. However, VSTS is flexible enough to accept our own provided custom build servers for execution of our build definitions.

When queuing a build definition, the name of an agent queue on which it should be queued should be provided. A queue is tied and mapped to an agent pool and any free agent in the pool that meets the capabilities needed by the build definition is free or available, eventually picks up the request and executes it.

Agents, agent pools and agent queues

The relationship between build servers, agents, agent pools and agent queues is shown in Figure 9.

Multiple agents can be installed on a build server.

Each of these agent belongs to a single agent pool. An agent cannot belong to multiple agent pools. An agent pool consists of multiple agents.

An agent queue is mapped to a single agent pool. A queue cannot be mapped to multiple agent pools however a single agent pool can be used by multiple agent queues.

Build Definition Configuration

The build management control panel can be reached through the Build link on top navigation bar. Clicking on the "+" button would start new build definition wizard. Figure 10 shows the "Create New Build definition" wizard.

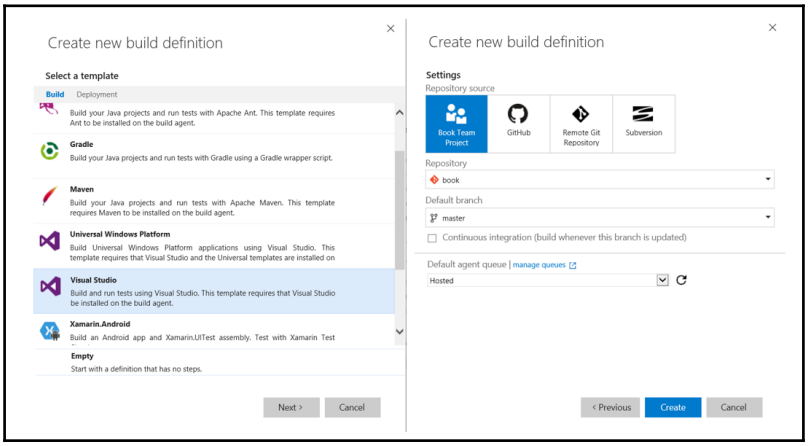

Figure 10: Creating Build Definition

Selecting a build template, providing appropriate repository name, branch name and agent queue information will create a build definition in draft mode with few activities in it. The build definition control panel is shown in Figure 11.

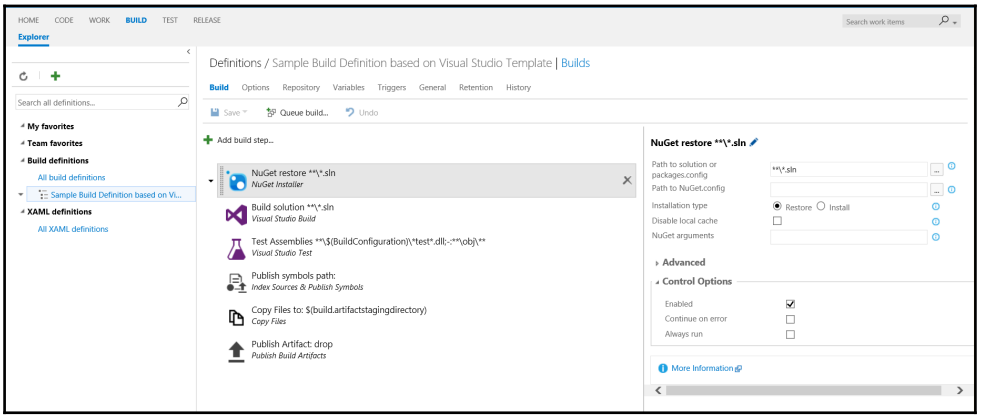

Figure 11: Build Control Panel

There are two levels of configuration for each build definition. We will look at them briefly

- **Build definition configuration –** These configurations effect the entire build definition and execution. It also provides the options to save and delete the build definition from any tab.
    1. **Build tab –** The build tab lists every activity that comprises the build definition. Tasks configuration for every task can be done from this tab. Refer to figure 11 for the same.
    2. **Options tab –** This tab is used to build for multiple configuration like debug, release or any other configuration defined in the project. Selecting "Multi-Configuration" shows further options. We can provide comma separated configuration names, select whether each of these configurations should run in parallel or in sequence and select whether these configurations should stop if an error occurs.

        It also allows to create a new work item (Bug, Epic, feature, Task, test case, user story, issue) when the build fails by checking the "Create a work item on failure" checkbox. The same is shown in Figure 12

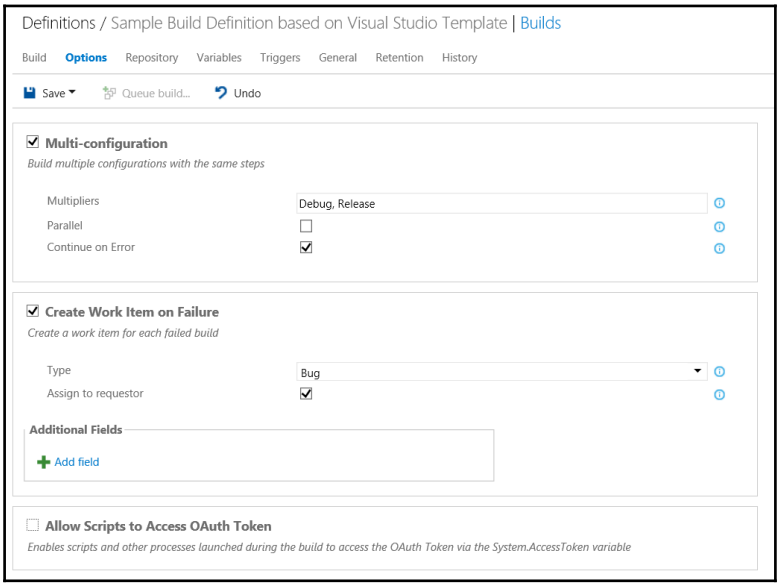

Figure 12: Options tab of Build definition

3. **Repository –** The options shown here are different based on the type of repository. For GIT related repository, information about the repository and its branch in the project should be provided. Clean option removes untracked files and branches from the repository. Label helps in providing labeling the source code files and version them for easy identification. For reporting of build status, it should be checked and if code is reused from other repository then submodule should also be checked out. Figure 13 depicts the same

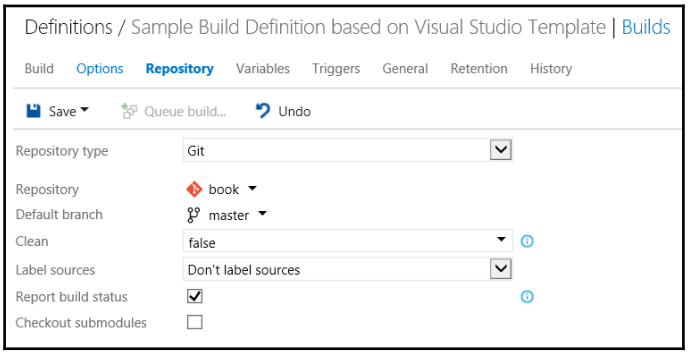

Figure 13: Repository tab of build definition

4. **Variables –** Variables help in making the build definition generic and reusable. It helps in removing hard coding and changes behavior of build definition while execution. They help in using the same value at multiple places within the same definition. There is a set of pre-defined variables available and user-defined variable can be defined. Variables have a name and a value. Variables can also be encrypted by using the secret lock button. Moreover, "allow at queue time" enables the available of the same variable during manual queuing providing an opportunity to change its value before execution. Figure 14 shows the variables tab in a build definition.

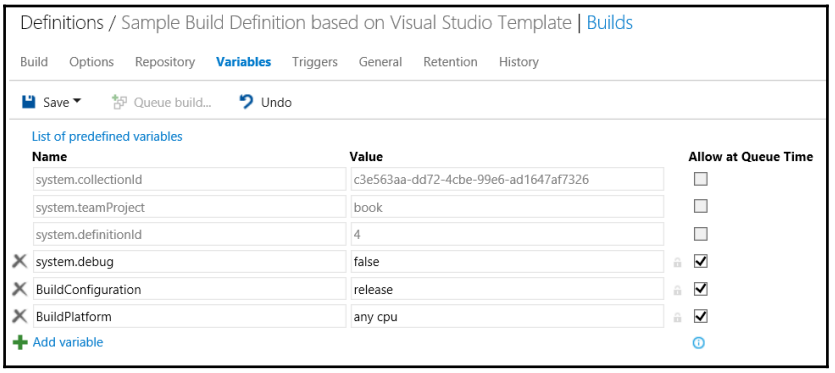

Figure 14: Variables tab of build definition

5. **Triggers –** Build can be initiated manually as a scheduled activity or/and as continuous integration. When Continuous integration is chosen, any commit or check-in of code will initiate the build process by queuing it to agent queue. Figure 15 shows the Triggers tab of build definition.

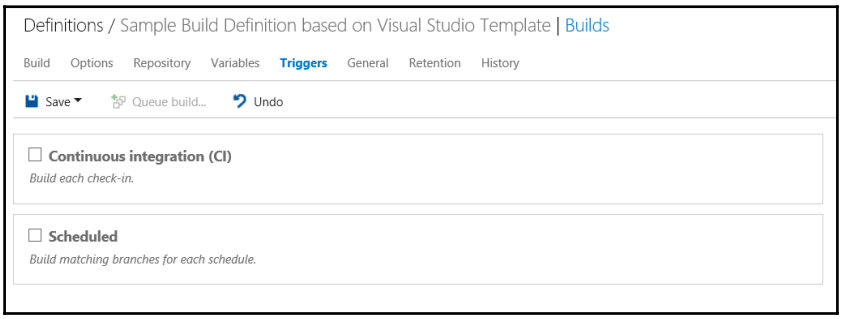

Figure 15:Triggers tab of Build definition

6. **General –** This tab allows choosing the queue for submission of our build definition. If the build needs access across projects in the account, "project collection" should be chosen as authorization scope else "current project" should be chosen. Description provides more information about the build definition. Every build is given a unique number and if we want more useful names for the build, "Build number format" can be used to define the same. The demands section lists the capabilities a build agent must possess for successful execution of the definition. Build will not execute if the demands are not satisfied. "Build timeout in minutes" provide control to VSTS to cancel the build execution if it does not finish within the given time. Figure 16 shows the general tab of build definition.

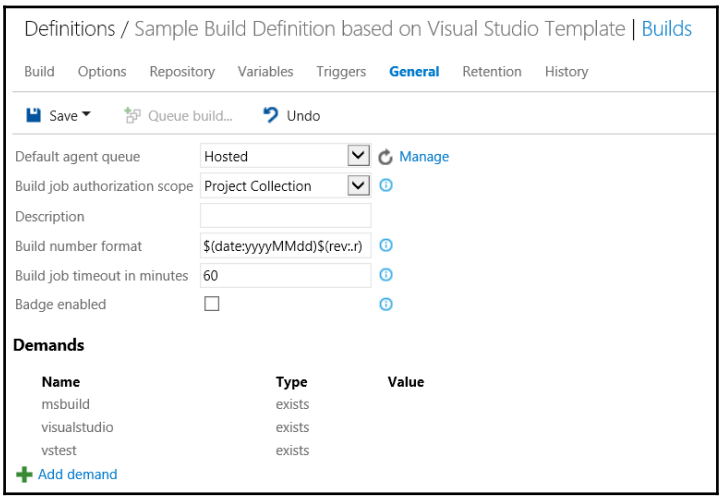

Figure 16: General tab of build definition

7. **Retention –** VSTS stores logs and other information for every build execution. This tab allows in setting the number of days for retaining the build logs. Figure 17 shows the retention tab

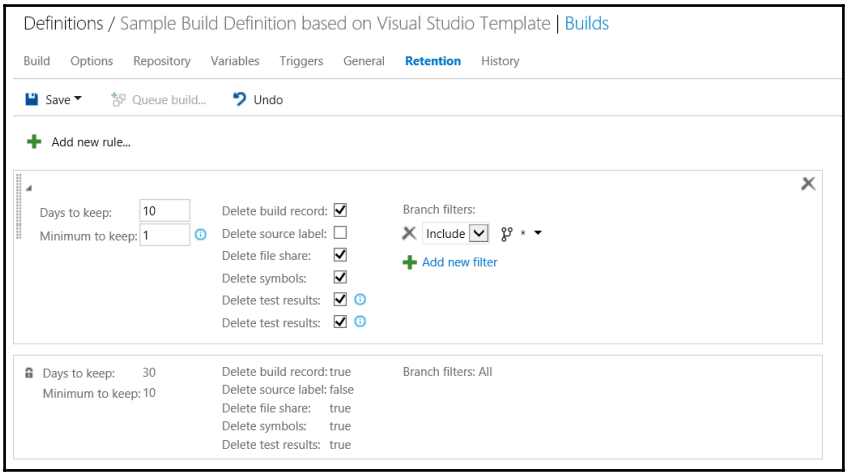

Figure 17: Retention tab of build definition

8. **History –** VSTS maintains the history of changes made to the build definition. This was not possible in earlier versions of VSTS. This provides version control for the build definition itself and allows for reverting back to previous version, comparison between multiple versions of same definition.

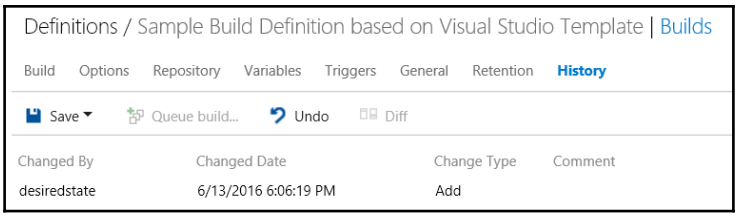

Figure 18: History tab of build definition

There is further configuration related to build configuration available from the build definition context menu. Important among them is the "security" configuration. "Security" configuration allows providing contributor, administrator and authoring permissions to users and groups. This is shown in Figure 19.

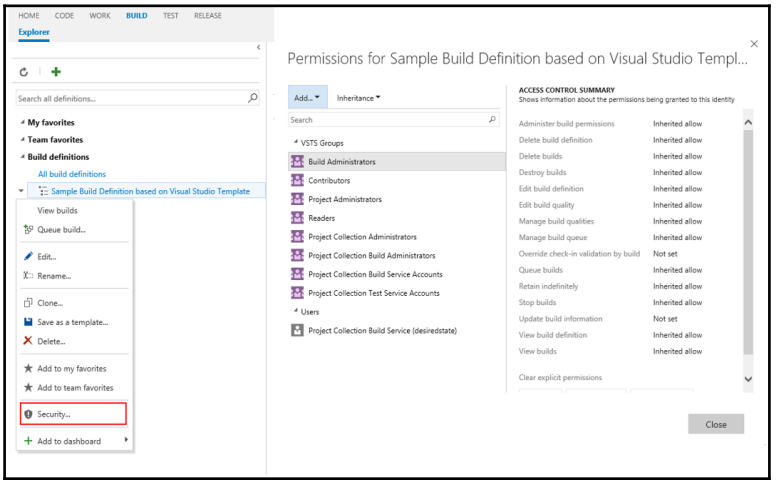

Figure 19: Build definition Security configuration

"Queue build" button helps in enlisted a build definition to a queue and start executing them.

- Individual tasks configuration

  Each task has its own set of configuration for it to work upon. There are different requirements for different tasks. Figure 20 shows the "Build solution" task. It is responsible for compiling every project within solution. It accepts the solution file path and multiple configurations related to NuGet. These include path to NuGet packages, whether to restore or install NuGet packages, disabling NuGet cache, NuGet arguments. The advance section accepts the path to the NuGet.exe.

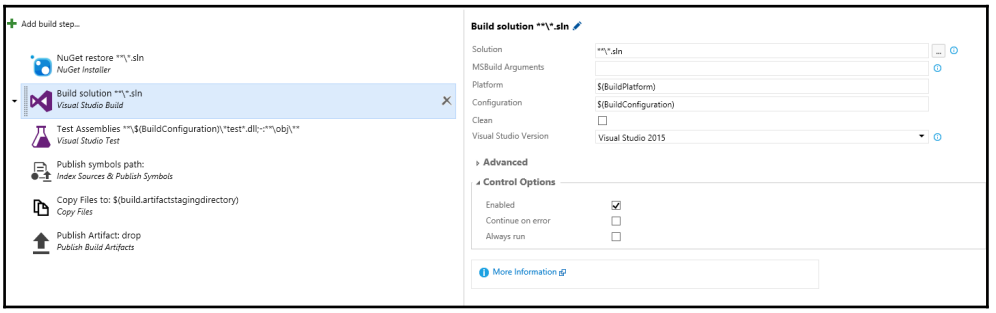

Figure 20: Build solution task in build definition

"Control Options" section has additional configuration that controls the entire task. These configurations are common to all the tasks available in VSTS build and release definition. "Control Option" is shown in Figure 20.

1. "Enabled" check refers to whether this task participates build definition execution. It is not executed if the check is removed. By default, it is in checked state.

2. A build execution fails when any tasks within it fails. "Continue on error" check helps in change this setting and allow continue executing the build even if this tasks fails.

3. "Always run" ensures that this task always execute even if there are build failures from other tasks.

Release management service

Now, it's time to look how VSTS can help in continuous deployment and continuous delivery. We already know by now, continuous deployment refers to deployment of application on multiple environments including pre-production and production environments through automation. This involves both provisioning and configuration of environments and application.

Release management service helps in automating the deployment and configuration to multiple environments. It helps in executing validation tests like functional tests on these environments. It is release management services that helps in implementing both continuous deployment and delivery. Similar to source code and build management, release services are scoped at project level. For each project, VSTS allows for creation of multiple release definitions.

Each release definition consists of multiple environment definition, each environment representing a deployment target. Test, authoring and production are example of environments. Release management executes these environment definitions and each environment can be configured to run in parallel or in sequence after a prior environment execution. Each environment definition would typically consist of provisioning of multiple servers, configuring them, deploy and configure application components on these servers and validating the application. Although the configuration of environments within a release definition could be different, however from DevOps perspective, they should be similar.

Each environment definition consists of multiple activities that run in sequence one after another like a pipeline. Each activity is responsible for executing a single task within the overall environment pipeline. Figure 21 shows a release definition example consisting of two environments authoring and production with each having multiple activities in its pipeline. Each contains the same tasks however their configuration is different for each environment.

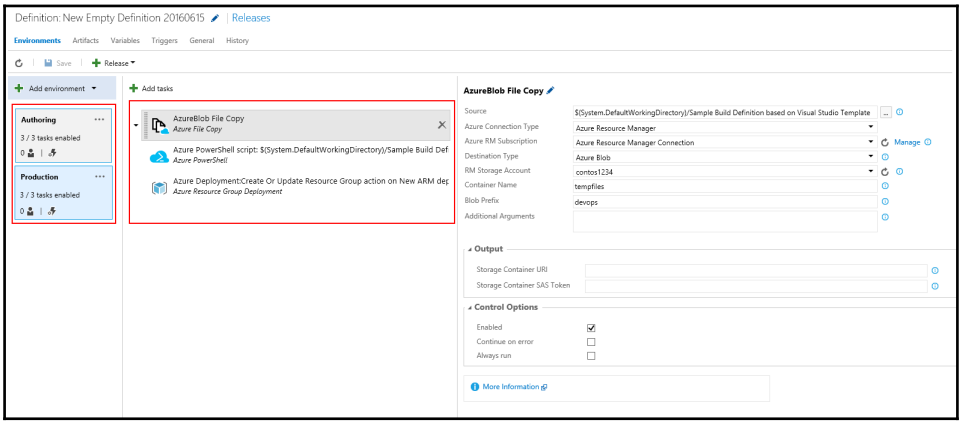

Figure 21: Release definition with two environments

Tasks available to both build and release definitions are common in VSTS. Figure 8 in this chapter shows the rich set of tasks available for build and release definition.

Execution of a release definition results in execution of each environment definition in a pipeline (sequential or parallel) and each environment definition in turn executes its tasks in a sequence pipeline.

The release pipeline gets its inputs in the form of artifacts. One of the final steps in continuous integration is to drop the generated package to s specified location. Release management considers the output from a build execution as an artifact. It also considers files stored in GIT and TFVC repositories as an artifact.

Executing Release Definitions

Release definitions needs release environment to be executed. The tasks in a release and environment definition has pre-requisites and those should be available on the release servers. VSTS provides in-built release servers out of the box with pre-requisites installed on them. They are available to every account in VSTS for executing release definitions. However, at the same time, VSTS is flexible enough to accept our own provided custom release servers for execution of release definitions.

Release architecture

The release management architecture is same as that of build management. Figure 9 depicts the build management architecture and is same for release management. Every release server should have release agent installed and running on them. Release agents are windows services that are configured to interact and work with VSTS release service. The agents are grouped together to for agent pools. VSTS provides a default agent pool named "hosted". Every VSTS provided servers are part of hosted agent pool.

Each release definition is configured to use an agent queue. Executing a release definition in turn executes the environment definitions. Each environment definition can be configured to use a particular Agent queue overriding the release level agent queue configuration. If not overridden, each environment uses the queue information from release definition. For each environment a request is queued on the agent queue. An agent queue is formed at project collection level and multiple queues can be created. When queuing an environment definition, VSTS expects the artifact that should be used within its pipeline. A queue is tied and mapped to an agent pool and any free agent in the pool that meets the capabilities needed by the environment definition is free or available, eventually picks up the request and starts executing it.

Agents, agent pools and agent queues

Refer to image Figure 9 in this chapter that depicts the relationship between release agents, pools and queues.

Release Definition Configuration

The release management control panel can be reached through the release link on top navigation bar. Clicking on the "+" button would initiate creation of a new release definition.

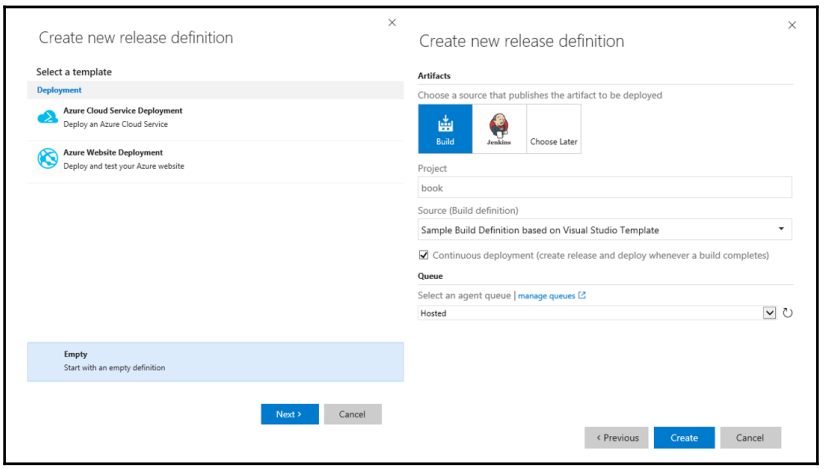

Figure 22:Creating a new release definition

Figure 22 shows the "Create New release definition" User interface. Selecting a release template, providing source artifacts and agent queue information will create a release definition in draft mode. The queue agent selected here will become the default queue agent for every environment within the release definition. If "Empty" template is chosen, VSTS will create an environment named "environment1" with no tasks in it. The same release definition is shown in Figure 23.

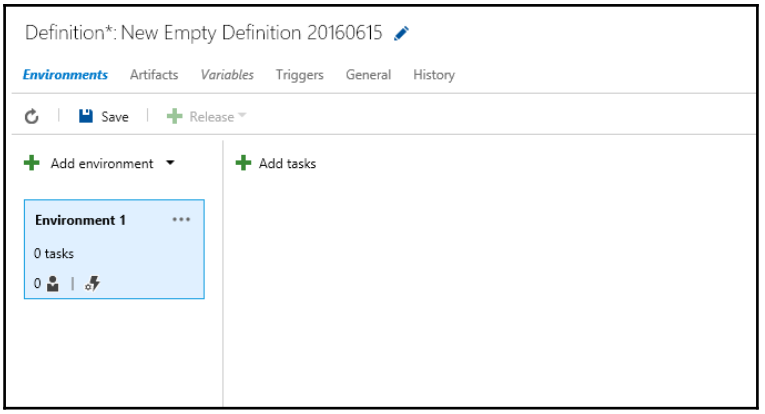

Figure 23: empty release definition

There are three levels of configuration for each release definition. We will look at them briefly

- **Release definition configuration –** These configuration effects the entire release definition and execution. It also provides the options to save the release definition from any tab. Deletion of a release definition is available from the release definition context menu as shown in Figure 18.

    1. **Environments tab –** The environment tab lists every environment available in the release definition. It also comprises of all the tasks that are part of environment definition. Individual task configuration can be done from this tab. This is already shown in Figure 23.

    2. **Artifacts tab –** This tab helps in configuring the link to the artifacts. The artifacts can come from a build execution or from a repository. Build artifacts can be chosen from current project only while repository artifacts can choose any repository in any project within the VSTS account. The artifacts user interface is shown in Figure 24.

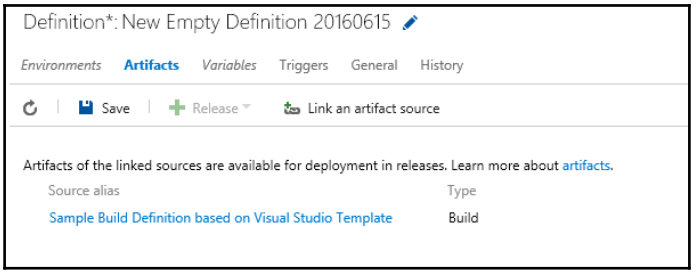

Figure 24: Artifacts tab in release definition

    3. **Variables –** variables help in making the release definition generic and reusable. It helps in removing hard coding and changes behavior of release definition while execution. They also help in using the same value at multiple places within the same definition. There are a set of pre-defined variables available and user defined variable can be defined. variables have a name and value. Variables can also be encrypted by using the secret lock button. These are release level variables. There is another set of variable known as “environment variables” that are defined at environment level. The same is shown in Figure 25. We will look at them when we talk about environment level configuration in next section.

Figure 25: Variables tab in release definition

4. **Triggers –** Release can be initiated manually as a scheduled activity or/and as continuous deployment. If Continuous deployment is chosen as an option, availability of any new build version or any new commit/check-in of code will start the release process by queuing it to agent queue. The artifacts configuration decides whether to use build output or new commits in code repository for initiating the release. Build number must be chosen manually as an artifact, if release is manually initiated. The same is shown in Figure 26. Trigger configuration is also available at environment level and we will look into it when we discuss environment specific configuration in next section.

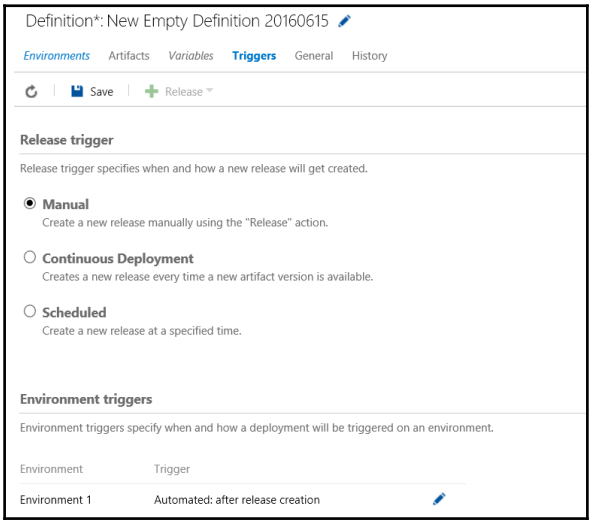

Figure 26: Triggers tab in release definition

5. **General –** This tab allows configuring the release name. Every release is given a unique number and "Release name format" helps in providing more useful names for the releases. VSTS stores logs and other information for every release execution. This tab allows in setting the number of days for retaining the release logs. The settings for tab is shown in Figure 27.

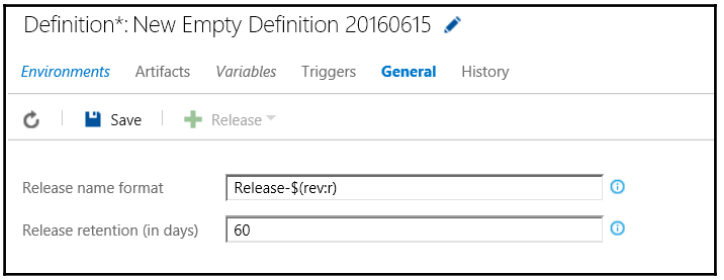

Figure 27: General tab in release definition

6. **History –** VSTS maintains the history of changes made to the release definition. This was not possible in earlier versions of VSTS. This provides version control for the release definition itself and allows for reverting back to previous version, comparison between multiple versions of same definition. This history details are shown in Figure 28.

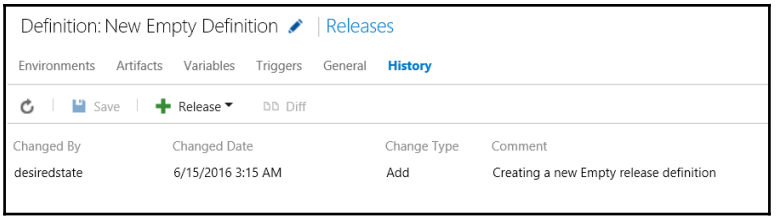

Figure 28: History tab in release definition

There is further configuration related to release configuration available from the release definition context menu. Important among them is the "security" configuration. "Security" configuration allows providing contributor, administrator and authoring permissions to different team members. The steps for security configuration for release definition is same as that of build

definition. Refer to build definition security section for same in this chapter.

Clicking on Release button and then on “create release” item creates a new release, enlists the environments on agent queues and starts executing them.

- **Environment configuration –** These configuration effects an individual environment definition and execution. Each environment has a button with symbol “...”. Clicking on it and then on any one of the menu item – “Assign Approvers”, “Agent Queues”, “Configure variables” and “deployment conditions” will show the user interface for configuration of that environment. This is shown in Figure 29.

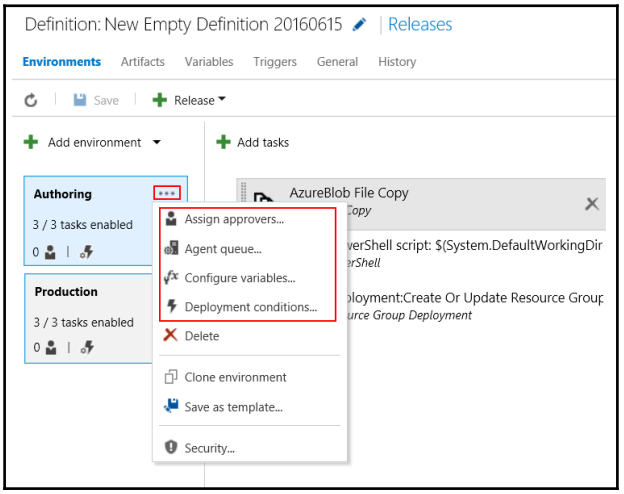

Figure 29: Environment context menu

Deletion of an environment definition is available from the its context menu as shown in Figure 30.

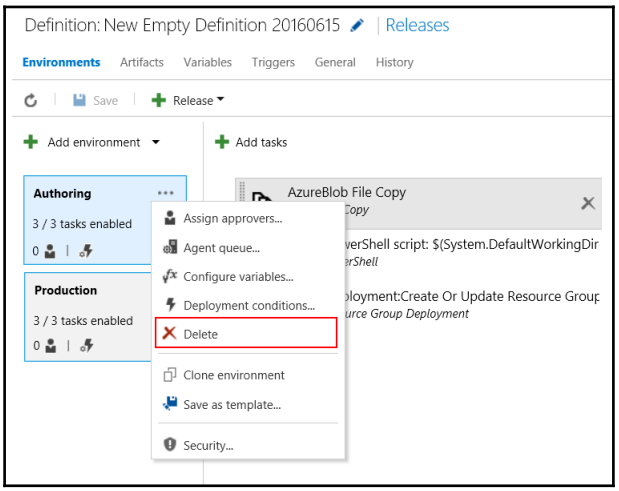

Figure 30: Deleting an environment definition

1. **Approvals tab –** Approvals plays a pivotal role in release management because it effects multiple environment that are strategic and important to application and organization. Approvals can be sought before execution of an environment definition from a user/group or approvals can be configured to be approved automatically. There are both pre-deployment and post-deployment approvals for an environment. This is shown in Figure 31.

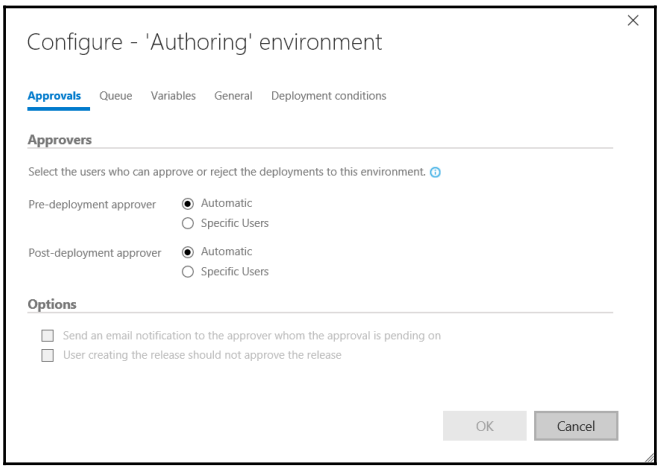

Figure 31: Approvals tab in environment definition

2. **Queue tab –** This tab can override the default agent queue. The demands section lists the capabilities a release agent must have for successful execution of the definition. Release will not execute if the demands are not satisfied. The queue configuration is shown in Figure 32.

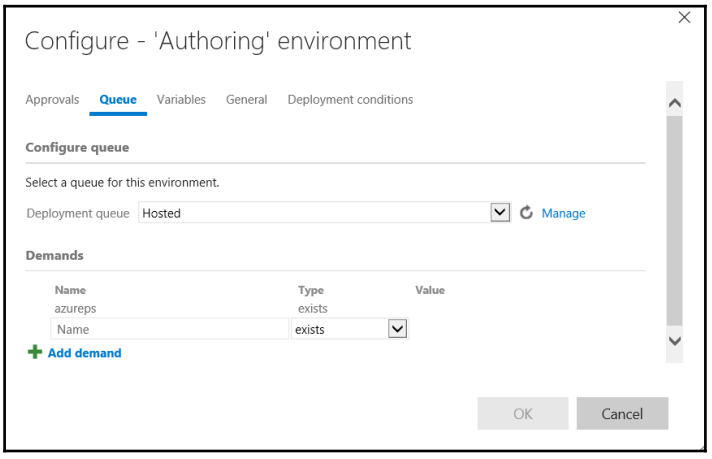

Figure 32: Queue configuration in environment definition

3. **Variables –** The concept of variables is same as that discussed during release configuration. Environment also provides the same set of pre-defined variables as provided at the release level and user defined variable can be defined. The release level variables are accessible at environment level and can also be overridden here by declaring variable with same name. The variables are scope within the current environment only. This is shown in Figure 33.

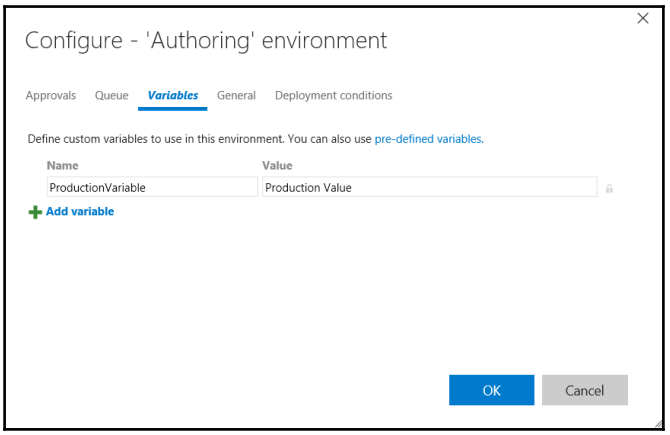

Figure 33: Variable configuration in environment definition

4. **General –** This tab allows configuration of email notification and also providing a name for the owner of the environment. “Skip artifacts download” ensures that the artifacts are not downloaded to the agent before starting the deployment. “Deployment timeout in minutes” provide control to VSTS to cancel the environment execution if it does not finish within the provided time. This is shown in Figure 34.

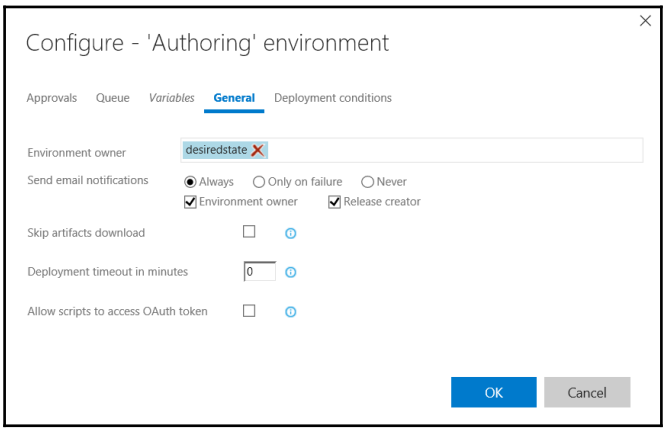

Figure 34: General configuration in environment definition

5. **Deployment conditions –** Environment release can be initiated manually. It can be triggered automatically after creation of release from release definition as a scheduled activity or/and as continuous deployment. When Continuous deployment is chosen, availability of any new version from build output or any commit or check-in of code will start the release process by queuing it to agent queue. The artifacts configuration decides whether to use output from build execution or new commits in code repository for initiating the release. If environment definition is executed manually, the artifact must be chosen along with it. This is shown in Figure 35.

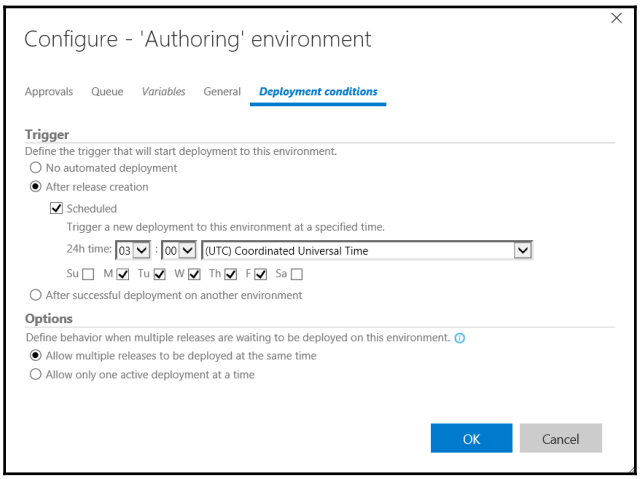

Figure 35: Deployment configuration in environment definition

- Individual tasks configuration

  Release definition have the same tasks available for its configuration as that available to build definition. We have already covered this section while discussing build tasks configuration.

Setting up Cloud Environment

Azure and VSTS accounts are crucial for implementing the DevOps processes. In this chapter, we will show the steps to create both Azure and VSTS accounts and in chapter 5, we will set up the development environment for our application.

Visual Studio Team Services

The primary pre-requisite for creating and account with VSTS is to have a Microsoft account. Microsoft account was earlier known as Live account. This is a free account that can be setup through https://signup.live.com for accessing Microsoft services like Skype, OneDrive, outlook.com, Hotmail.com and more. It is a Hotmail or outlook.com account or any other live account.

Another way to create a VSTS account is to have a “work or school account” which refers to

an enterprise and its email accounts.

One account, either a Microsoft account or "Work or school account" is necessary to create a VSTS account. After provisioning these accounts, browsing through https://go.microsoft.com/fwlink/?LinkId=307137&clcid=0x409 will start a wizard to create VSTS account. It will ask to login with your Microsoft or Work account. After login, pick a unique name through which your VSTS is identified. This is shown in Figure 36. You can also select the type of repository – Git or TFVC. By clicking on change details link, select a preferred region and the process types (CMMI, SCRUM and Agile) to manage the work. This is shown in Figure 37.

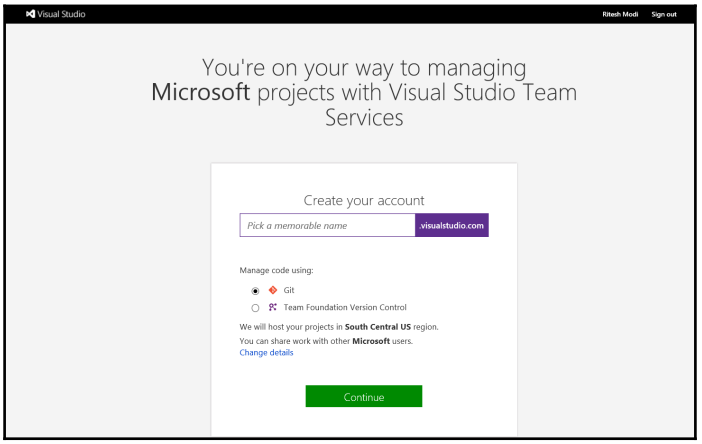

Figure 36: Creating a VSTS account

You can also select the type of repository – Git or TFVC. By clicking on change details link, it is possible to select a preferred region and also the type of process (CMMI, SCRUM and Agile) to manage the work. Clicking on continue button will take few seconds to create an account. This is shown in Figure 37. This will also create a new project named "MyFirstProject" by default with provided configuration as shown in Figure 37.

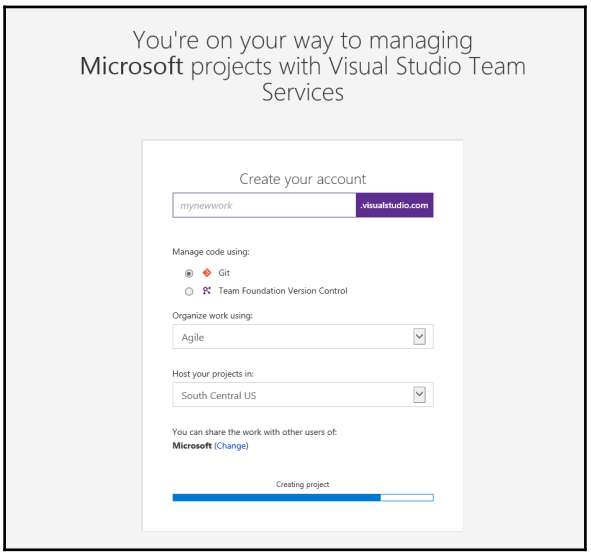

Figure 37: Option while creating VSTS account

Azure Account

The primary pre-requisite for creating and account with Microsoft Azure is to either have a Microsoft or a student/work account. We already saw in last section about way to create a Microsoft account. After having a valid account, navigate to https://account.windowsazure.com from browser. Click on the “Sign-in” button on extreme top right of the page. This is shown in Figure 38.

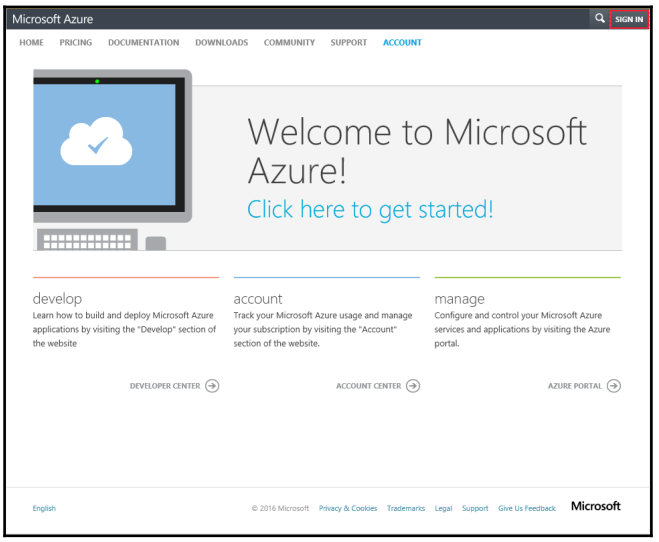

Figure 38: Azure start Page for creating a subscription

This will navigate to login page for your Microsoft or work account. I have an outlook account and am using the same for obtaining a subscription and a tenant from Azure. This is shown in Figure 39.

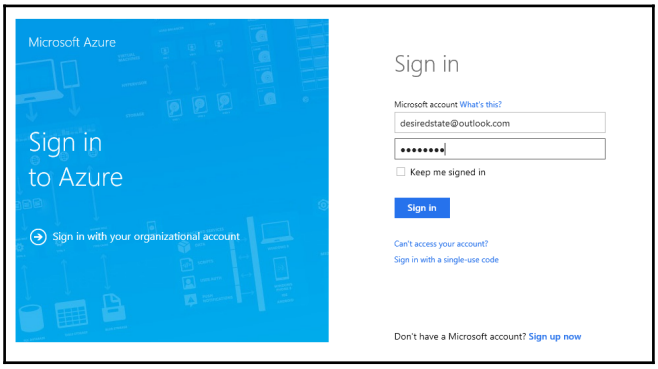

Figure 39: Azure login screen

Logging into Azure will take to screen shown in Figure 40. Click on “Sign up” link.

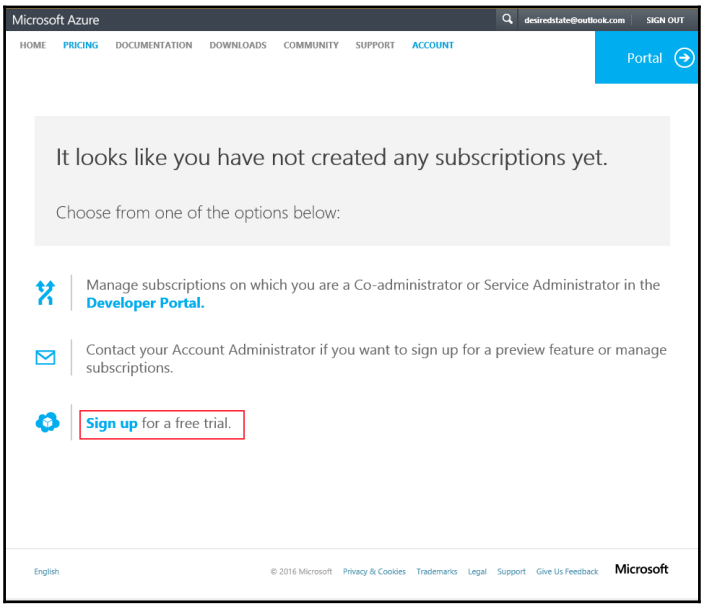

Figure 40: Signup with Azure

It will show details about your account, ask for validating through your phone, verification of credit card. This is shown in Figure 41. After accepting the agreement terms, click on "Sign up" to create a tenant and subscription in Azure.

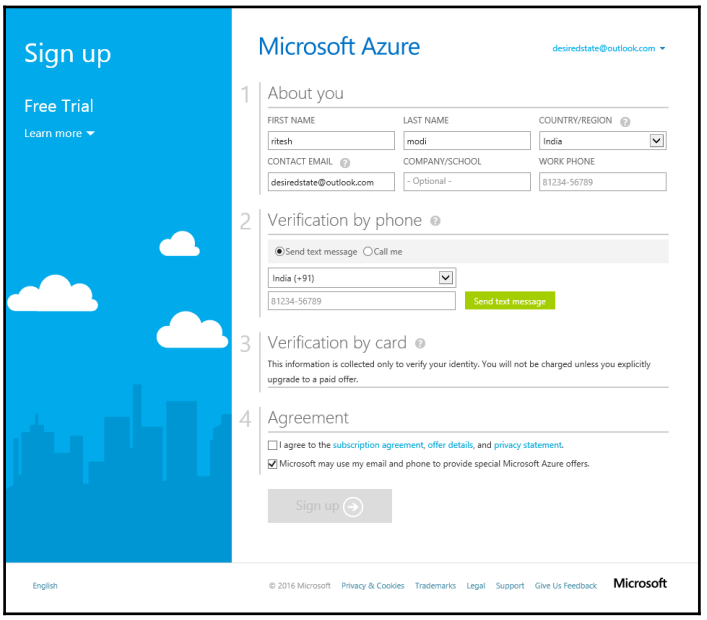

Figure 41: Form to create, validate subscription in Azure

Summary

We covered a lot of ground in this chapter. Earlier chapter introduced DevOps concepts and this chapter is about mapping technology to those concepts. In this chapter, we saw the impetus DevOps can get from technology. We understood cloud computing and different services provided by Cloud providers. From there, we understood the benefits Windows Server 2016 server brings to the DevOps practices and how Windows 2016 server makes DevOps easier and faster with its native tools and features. Towards the end, we explored VSTS that forms the core for DevOps practices by implementing Continuous integration, Continuous deployment and delivery. You should now have a good grasp of tools and technology used to implement DevOps. We created an account in VSTS for our DevOps process and created a subscription on Azure used for hosting out platform and application.

From next chapter onwards, we will get into details of each of these technology and use them in practice.

3

DevOps Automation Primer

Azure was launched in early 2010 with Azure Service Management (ASM) as its base technology platform for provisioning, organizing and managing IaaS and PaaS services. During Build 2014 event, Microsoft introduced a new Azure technology platform, Azure Resource Manager (ARM). There are inherent issues with Azure Service Management in terms of performance, concurrency, extensibility and scalability of services. It was becoming difficult for Microsoft to introduce newer, consistent and scalable services because of the way ASM was designed and architected. Azure Resource manager was introduced to overcome these challenges and provide an architecture that is extensible, scalable and provide additional advance features not available with ASM.

Before 2014, System center configuration manager (SCCM) was the prime configuration management software from Microsoft and with cloud gaining popularity, there was a need of a light weight configuration management platform that could easily scale and run everywhere including cloud and on-premise. Desired State Configuration (DSC) was launched in 2014 as part of Windows Management Framework (WMF) 4.0 and has become stable and robust with the release of WMF 5.0. Windows Serve 2016 and Windows 10 comes with WMF 5.0 installed out of box. DSC is lightweight configuration management platform, capable of running and configuring multiple operating systems, services and environments. Powershell has been for long in the windows world. It is the de-facto standard and used ubiquitous for administration and management of infrastructure, environment and services.

In this chapter, I introduce Azure resource manager, describe its concepts, benefits and advantages, cover its architecture and show its features. The chapter continues with Azure Resource manager templates. Templates is a deployment feature of ARM and enables Infrastructure as code and DevOps. I will show templates are defined and authored, its features and components like linking, dependencies, expressions, monitoring and auditing deployments and tools for authoring them.

Next, Powershell will be introduced. It is a scripting language and command line shell used for automation, administration and management of servers, services and environments.

Another Powershell based utility Pester will be discussed next. Pester is unit testing tool of powershell scripts. It helps in validating infrastructure and environments through assertion, comparing desired with actual output. The chapter ends with how all these technologies comes together and helps in enabling DevOps. DSC is based on Powershell and Azure templates uses both DSC and Powershell for provisioning and management of Azure resources. I will try to cover these technologies as much as possible in this chapter, however, because this book is on DevOps, these topics will not be covered in complete detail and each demands a book by themselves.

Next section will introduce DSC, describe its different architecture and concepts, show its configuration features and managing environments using dependencies, partial configuration, Powershell cmdlets and introduce DSC resources.

Azure Resource Manager

ARM is successor of ASM. Although both the platforms are operational and available as of writing this chapter, the direction from Microsoft is to use ARM as a platform for all future deployments.

ARM and ASM

ASM has inherent constraints and some of the major ones are discussed here.

- ASM deployments are slow and blocking. Operations are blocked if an earlier operation is already in progress.
- Concurrency is a challenge in ASM. It is not possible to execute multiple transactions successfully in parallel. The operations in ASM are linear and executed one after another. Either the parallel operation errors out or will get blocked.
- Resources in ASM are provisioned and managed in isolation to each other. There is no relation between ASM resources. Grouping of services and resources, configuring them together is not possible.
- Cloud Services is the unit of deployment in ASM which are reliant on Affinity groups and not scalable due to its design and architecture.
- Granular and discreet roles and permissions cannot be assigned to resources in ASM. Users are either service administrators or co-administrators in the subscription. They either get full control on resources or do not have access to them at all.
- ASM provides no deployment support. Deployments are either manual or resorts

to writing procedural scripts in Powershell or .net for deploying environments.
- ASM API's were not consistent between resources.

ARM advantages

ARM provides distinct advantages and benefits over ASM.

- **Grouping:** ARM allows for grouping of resources together in a logical container. These resource can be managed together as a group. It is easier to identify related and depended resources.
- **Common Lifecycle**: Resources in a group have the same lifecycle. These resources can evolve and managed together as a unit.
- **Role based Access control:** Granular roles and permissions can be assigned to resources providing discreet access to users. Users can have only those rights that are assigned to them.
- **Deployment support:** ARM provides deployment support in terms of templates enabling DevOps and Infrastructure as code. The deployments are faster, consistent and predictable.
- **Superior technology:** Cost and billing of resources can be managed as a unit. Each Resource groups can provide their usage and cost information.
- **Manageability:** ARM provides advance features like security, monitoring, auditing and tagging features for better manageability of resources. Resources can be queried based on tags. Tags also provide cost and billing information for resources tagged similarly.
- **Migration:** Easier migration and update of resources within as well as across resource groups.

ARM Concepts

ARM is based on concepts related to resource providers and resource consumers. Azure provides services through multiple resource providers that are consumed and deployed in groups.

Resource Providers

These are services responsible for providing resource types to Azure Resource Manager. Resource types are grouped into namespaces and resource providers are identified through

these namespaces. They are responsible for deploying and managing the resources and resource types. For example, a virtual machine resource type is provided by resource provider identified by Microsoft.Compute namespace. Each Resource provider have multiple versions as they evolve over a period of time. The versions are based on date provided by Microsoft. Resource providers are not available to a subscription out of box. They must be registered with a subscription before their resources can be deployed on it. Although some of the resource providers are automatically registered in a subscription, other needs to be explicitly registered. Also, not all resources providers are available at every Azure region. Resource providers are capabilities that should be checked for their availability in a region before using its resources for deployment in that region.

Resource Types

These are actual resource specification defining its public api interface and implementation. It implements the working and operations supported by the resource. Similar to resource providers, Resource types also evolves over time with regard to their internal implementation and have multiple versions for its schema and public api interface. The versions names are based on dates on which they are released by Microsoft as preview or general availability (GA). The resource types become available to a subscription after a resource provider is registered to it. Also, not every resource type is available at every Azure region. Their available is based on availability of resource provider in a region. In effect, availability of a resource is dependent on availability and registration of a resource provider in an azure region and must support the api version needed for provisioning it.

Resource Groups

Resource groups are unit of deployment in ARM. They are containers grouping multiple resource instances in a security and management boundary. A resource group is uniquely named in a subscription. Resources can be provisioned on different Azure regions yet belong to same resource group. It provides additional services to all resources within it. Resource group provides metadata services like tagging enabling categorization of resources, policy based management of resources, role based access control, protection of resources from accidental deletion or update and more. As mentioned before, they are security boundary and users not having access to resource group cannot access resources contained in it. Every resource instance needs a resource group and without them resources cannot be deployed.

Resource/ Resource Instances

Resources are created from resource types and should be unique within a resource group. The uniqueness is defined by the name of resource and its type together. In object orientation, they can be referred as objects while resource types can be referred as class. The services are consumed through the operations supported and implemented by resource instances. They define properties that should be configured before usage. Some are mandatory properties while others are optional. They inherit the security and access configuration from its parent resource group. These inherited permissions and role assignments can be overridden for each resource. The resources can be locked such that some of its operation can be blocked and not made available to roles, users and groups even though they have access to it. They can be tagged for easy discoverability and manageability.

Azure Resource Manager

Azure Resource manager is the technology platform and orchestration service from Microsoft that ties up all components discussed earlier. It brings Azure Resource providers, resources and resource groups together to form a cohesive cloud platform. It helps in registration of resource providers to subscriptions and regions, it makes the resource types available to resource groups, make the resource and resource api's accessible to portal and other clients, authenticates access to resources. It also enables features like tagging, authentication, role based access control, resource locking and policy enforcement for subscriptions and its resource groups. It provides the same deployment and management experience whether through portal or client based tools like Powershell or command line interface.

Azure Resource Manager Architecture

Figure 1 shows the architecture of Azure resource manager and its components. As shown in the figure, A subscription comprises of multiple resource groups, each resource group contains resource instances that are created from resource types available in resource provider.

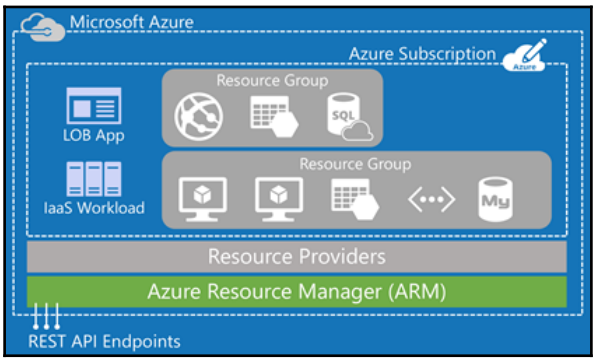

Figure 1: Azure Resource Manager Architecture

Azure Resource Manager Features

Following are some of the major features provided by Azure resource manager.

Role Based Access Control(RBAC)

Azure Active Directory(AAD) authenticates users to provide access to subscriptions, resource groups and resources. ARM implements oAuth and RBAC in platform itself enabling authorization and access control to resources, Resource groups and subscriptions based on roles assigned to user or group. A permission defines access to operations of a resource. These permissions are could be allow or deny on the resource. A role definition comprises of these permissions. Roles maps AAD users and groups to the permissions. Roles are subsequently assigned to a scope which refer to individual resource, group of resources in a resource group, resource group or subscription. The AAD identities (users and groups) added to a role gets access to the resource according to permissions defined in the role. ARM provides multiple roles out of box. It provides system roles like Owner, Contributor, Reader and more. It also provides resource based roles like SQL DB contributor, virtual machine contributor and more. ARM allows to create custom roles.

Tags

Tags are name value pairs that add additional information and metadata to a resource. Both resources and resource groups can be tagged with multiple tags. Tags helps in categorization of resources for better discoverability and manageability. Resources can be searched and identified with ease and quickly. Billing and cost information can be fetching for resources having the same tags applied. While this feature is provided by ARM, an IT administrator defines its usage and taxonomy with regard to resources and resource

groups. Taxonomy and tags for example can be defined based on departments, resource usage, location, projects or any other criteria deemed fit from cost, usage, billing and search perspective. These tags can then be applied to resources. Tags defined at Resource group level are not inherited by its resources.

Policies

Another security feature provided by ARM are policies. Custom policies can be created to control access to the resources. Policies are defined conventions and rules and those must be adhered to while interacting with resources and resource groups. Policy definition contains explicit denial of actions on resources or access to resources. By default, every access is allowed if it is not mentioned in the policy definition. These policy definitions are assigned to resource, resource group and subscriptions scope. It is important to note that these policies are not replacement or substitute to RBAC. In fact, they complement and works together with RBAC. Policies are evaluated after a user is authenticated by AAD and authorized by RBAC service. ARM provides JSON based policy definition language for defining policies. Some of the examples of policy definition are that it is must to tag every provisioned resource, resources can only to provisioned at limited set of Azure regions.

Locks

Subscriptions, Resource Groups and resources can be locked to prevent accidental deletion and updates by an authenticated user. Locks applied at higher level flows downstream to child resources. Locks applied at subscription level, locks every resource group and resources within it.

Multi-Region

Azure provides multiple region for provisioning and hosting of resources. ARM allows resources to be provisioning at multiple different location and yet reside within same resource group. A resource group can contain resources from different regions.

Idempotent

This feature ensures predictability, standardization and consistency in resource deployment by ensuring that every executed deployment will ensure the same state for resources and their configuration no matter the number of times it is executed.

Extensible

ARM architecture provides extensible architecture to allow creation and plugging of newer

resource providers and resource types into the platform.

Azure Resource Manager Templates

In earlier section, we witnessed deployment features like multi-service, multi-region, extensible, idempotent provided by ARM. ARM templates are the primary means of provisioning resources in ARM. ARM templates provides implementation support for ARM deployment features.

ARM templates provides a declarative model through which resources, their configuration, scripts and extensions are specified. ARM templates are based on JavaScript Object Notation (JSON) format. It uses the JSON syntax and conventions to declare and configure resources. JSON files are text based, human friendly and easily readable files. They can be stored in a source code repository and have version control on them. They are also means to represent infrastructure as code that can be used to provision resources in Azure resource group again and again predictably, consistently and uniformly. A template needs a resource group for deployment. It can only be deployed to a resource group and the resource group should exist before executing template deployment. A template is not capable of creating a resource group.

Templates provide the flexibility to be generic and modular in their design and implementation. Templates provides ability to accept parameters from users, declare internal variables, helps in defining dependencies between resources, links resources within same or different resource groups and also executes other templates. They also provide scripting language type expressions and functions that makes them dynamic and customizable at runtime.

Template Basics

A bare minimum template that actually does nothing is shown here.

```
{
   "$schema": "http://schema.management.azure.com/schemas/2015-01-
   01/deploymentTemplate.json#",
   "contentVersion": "1.0.0.0",
   "parameters": {
   },
   "variables": {
   },
   "resources": [
   ],
```

```
    "outputs": {
    }
}
```

A template has four important sections – parameters, variables, resources and outputs. In preceding template, there are no defined parameters, variables, resources and outputs. Resources is the only mandatory JSON section while the rest are optional. Schema element defines the uniform resource location (URI) that contains the model a template should be bound and adhere to. It contains definition for all elements that can be defined in a template. The model itself contains references all Azure resource schema that can be defined in a template. Schemas help in design time validation of the template. Each template has a contentVersion element. This element defines the version of the template. Templates version numbers are used when invoking and executing nested templates. Parameters, variables and outputs section are JSON objects while the resources section is a JSON object array that can contain multiple JSON objects each representing a resource.

Parameters

Parameters helps to create generic and customizable templates. Parameters are defined in templates whose values are provided by user as arguments as part of deployment. This encourages the use of the same template for multiple different environments like Dev, test, production and other types of environments. Multiple parameters can be defined in a template and let's look at a sample parameter definition containing two parameters – firstParameter and secondParameter. The first parameter is of type string, can hold any one of two allowed values, the default values is FirstValue, has maxLength and minLength as validators. Metadata helps in adding additional context and information. Description is added as part of firstParameter metadata. Similarly, secondParameter is of type int with validators on acceptable values, minimum values and maximum value.

```
"parameters": {
    "firstParameter": {
        "type": "string",
        "allowedValues": [
            "FirstValue",
            "SecondValue"
        ],
        "defaultValue": "FirstValue",
        "maxLength": 20,
        "minLength": 10,
        "metadata": {
            "description": "My first parameter"
        }
    },
    "secondParameter": {
```

```
        "type": "int",
        "allowedValues": [
          10,
          20
        ],
        "defaultValue": 10,
        "maxValue": 20,
        "minValue": 10,
        "metadata": {
          "description": "My second parameter"
        }
      }
    }
```

Templates provide attributes for defining parameters as shown in code. The explanation of these attributes are mentioned in table here.

Attribute Name	Data type	Mandatory	Description
type	string	Required	Valid data types are string, securestring, int, bool, object and array
defaultValue	Depends on data type	Optional	Default values are used if the value is not provided
allowedValues	array	Optional	Valid values can only be one of the provided values.
minValue	integer	Optional	Minimum value for int type parameter.
maxValue	Integer	Optional	maximum value for int type parameter.
minLength	integer	Optional	Minimum length for string or array type parameter.
maxLength	integer	Optional	Maximum length for string or array type parameter.
Metadata	object	Optional	Can be any valid JSON object

Variables

Variables are similar to parameters but the values are defined internal within the template itself and is not provided externally by a user. The value of a variable is part of variable declaration itself. Variables are declared once and should be unique within a template. They can be placed anywhere within a template where a JSON string is expected. They make templates dynamic, manageable and changes can be done to it easily. The value of variable

is substituted during deployment at all places it's used in the template. Variables are of JSON object data type. Let's look at a sample definition of a variable as shown here

```
"firstVariable": {
   "networkName": "FirstNetwork",
   "subnets": [
      {
         "subnetName": "FrontEnd",
         "subnetIPRange": "10.0.0.0/24"
      },
      {
         "subnetName": "backEnd",
         "subnetIPRange": "10.0.1.0/24"
      }
   ]
}
```

Resources

Resources are array types that can hold multiple resource declarations. Arrays in JSON are represented by square brackets '[]' and objects by wriggly brackets '{}'. Each resource is an object declaring its desired configuration. Each resource two types of properties – properties that provides information to ARM about name, type, version and location of resource and properties that configures the resource itself. The mandatory properties of a resource are

- **name** – It represents the name of the resource instance.
- **apiVersion** – It specifies the version of resource to be used for provisioning.
- **location** – Azure region of the resource
- **Type** – The resource provider namespace along with resource type name for creating a resource instance.

Each resource has its specific configuration requirements and they would differ from one resource to another. These configurations configure the resource and its inner working. Let's look at the way to define resources in a template. There are two resource declaration in the resources section of the template.

The first resource instance is named “storageaccount” provisioned at “West Europe” azure region based on resource provider “Microsoft.Storage” and resource type “storageAccounts” and its version “2015-06-15”.

The first resource instance is named “myPublicIPAddress” provisioned at “West Europe” azure region based on resource provider “Microsoft.Network” and resource type

“publicIPAddresses” and its version “2015-05-01-preview”. Both the resources have a properties element which describes the resource specific configurations. StorageAccounts resource type has dependency on accountType property which defines whether the storage account should be locally redundant, zone redundant, Geo redundant or read access geo redundant. Similarly, PublicIpAddresses resource type has dependency on allocation method which can be dynamic or static. DNS settings provide a DNS name to the Public IP address.

```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "name": "storageaccount",
    "apiVersion": "2015-06-15",
    "location": "West Europe",
    "properties": {
      "accountType": "Standard_LRS"
    }
  },
  {
    "apiVersion": "2015-05-01-preview",
    "type": "Microsoft.Network/publicIPAddresses",
    "name": "myPublicIPAddress",
    "location": "West Europe",
    "properties": {
      "publicIPAllocationMethod": "Static",
      "dnsSettings": {
        "domainNameLabel": "mypublicipaddress"
      }
    }
  }
]
```

Outputs

Outputs represents the return values from templates as the result of executing or deploying them. Outputs section can be customized to contain multiple objects each returning values. Each objects in outputs section has two properties – The type of return value and the value of return type. The type refers to datatypes which we saw in the previous section and value refers to actual data or object returned by the template. Output section is executed and returns value only if the template was executed and deployed successfully. Let's look at the way to define outputs in a template.

```
"outputs": {
  "myOutput": {
    "type": "string",
```

```
        "value": "Resource Group deployed successfully !!"
    }
}
```

In the code listing a single output “myOutput” is defined. It is of type string and will return the text on successful execution of template.

Expressions and functions

ARM extends JSON by adding additional features in terms of expressions and functions. These are not available in JSON out of box and Microsoft added them to make templates dynamic and customizable. Expressions and functions are evaluated at deployment and they help in adding scripting language semantics to templates. Expressions are defined using square brackets and can appear anywhere a JSON string is expected in template. The return value from an expression is always in JSON format.

There are numerous functions provided by ARM templates and can be categorized into string functions, numeric functions, array functions, deployment functions and resource functions.

Numeric functions help working with integers like adding, subtracting, dividing numbers.

String functions help working with string literals like concatenation of strings, splitting of strings into array, replacing a part of string, getting substrings from original strings and more.

Array functions help in working with array values in a template like concatenating two arrays, splitting an array, retrieving sub-elements of an array and more.

Deployment functions help in getting values from variables, parameters and more.

Resource functions help in working with resources like getting their ID, current location and subscription, getting properties of resources and more. For complete list of function available for templates are available at https://azure.microsoft.com/en-in/documentation/articles/resource-group-template-functions/#resource-functions

Nested Resources

Resources can be nested within other resources however, both the parent and the child resource should support nesting. Not all resources can be nested. Examples of nested resource is Powershell extensions resources within virtual machine parent resource.

Another example of nested resource are Service bus queues, replay and topics should be contained within a service bus namespace. Nesting means that the child resource is part of parent resource although the declaration of child resources in a template can be within the parent resource or outside of it. There is special naming consideration to be taken care off while declaring a child resources outside of parent resource. It is important to note that contained resources are dependent on parent resource and cannot exist without them.

Some examples of expressions and functions are shown for better understanding

- Concatenating two strings

```
"[ concat( 'String 1', 'string2' ) ]"
```

- Adding two numbers

A minimal template

Let's look at a complete ARM template consisting of parameters, variables, resources and output. This template also uses expressions and functions. The purpose of this template is to provision a Azure storage account. This template takes in one parameter “storageAccountName” of type string. Storage names must be of minimum three characters and cannot be more than 24 characters in length. Couple of variables “storageApiVersion” and “storageAccountType” are defined will valid values. Resources section declares a single resource. The value for resource name is derived from “storageAccountName” parameter. The resource provider is Microsoft.Storage and resource type is storageAccounts. The value for apiVersion is retrieved from “storageApiVersion” variable and value for storage account resource specific “accountType” property is retrieved from “storageAccountType” variable. The storage account is provisioned at the same location of Resource group itself. ARM provides resourceGroup function for retrieving the current resource group on which the deployment is in progress. Finally, outputs section outputs the status of successful execution of deployment.

```
{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-
  01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountName": {
      "type": "string",
      "minLength": 3,
      "maxLength": 24,
      "metadata": {
        "Description": "Storage account name"
```

```
            }
        }
    },
    "variables": {
        "storageApiVersion": "2015-06-15",
        "storageAccountType": "Standard_LRS"
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[parameters('storageAccountName')]",
            "apiVersion": "[variables('storageApiVersion')]",
            "location": "[resourceGroup().location]",
            "properties": {
                "accountType": "[variables('storageAccountType')]"
            }
        }
    ],
    "outputs": {
        "TemplateOutput": {
            "type": "string",
            "value": "[concat(parameters('storageAccountName'),' account was
            successfully created!!')]"
        }
    }
}
```

We will use this template in subsequent section for deployment to Azure Resource Group.

ARM Template Tools

Working with Arm templates requires tools for both authoring and deployment.

ARM templates are simple text based JSON files. They can be authored using any text based editor however, for faster and easy authoring, intellisense support for template and its resource configuration, Visual Studio Code or Visual Studio 2013/2015 can be used. Visual Studio provides rich interface, project template, intellisense and deployment script for templates. This book uses Visual Studio 2015 for authoring of all ARM templates.

Although templates can be authored manually through Azure portal, it is not recommended as it is error prone and time consuming to author templates from there.

Authoring tools

In this book, we will use visual studio 2015 for authoring ARM templates. The steps for creating a template are shown next.

1. The first step is to install visual studio 2015 community edition on the development box. The development environment contains Windows server 2016 technical preview operating system. This is because Windows 10 does not yet support windows container and Docker. Eventually, when Windows 10 starts supporting windows containers and Docker, it should be used as development platform. Visual studio community edition is available from https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx . Click on “Download community free” button to start downloading visual studio. This is shown here in figure 2.

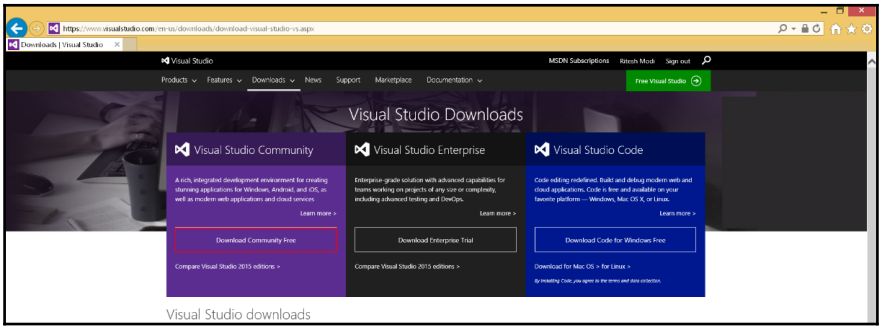

Figure 2: visual studio site for downloading community edition

A setup file is downloaded and executed. It will prompt a window asking for installation location and type of installation. Accept default and click on Install. This is shown in figure 3.

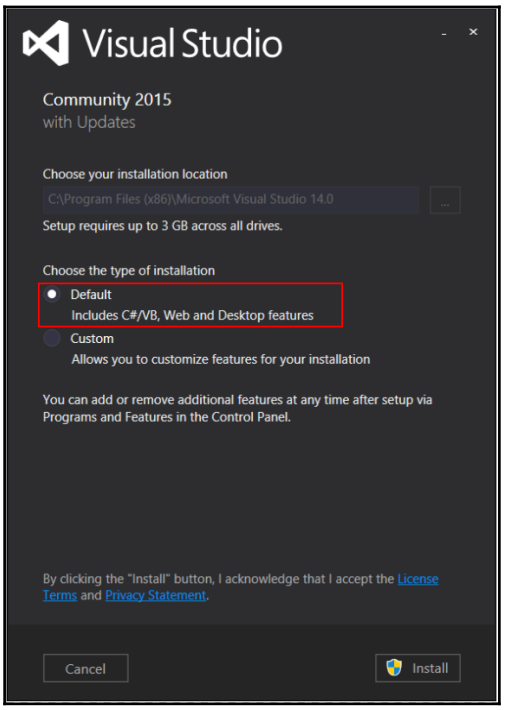

Figure 3: Installing visual studio 2015 community edition

2. After visual studio is installed, Azure tools for visual studio should be installed. The installer is available at https://azure.microsoft.com/en-us/tools/ shown in figure 4. click on button “Download Azure Tools for Visual Studio”.

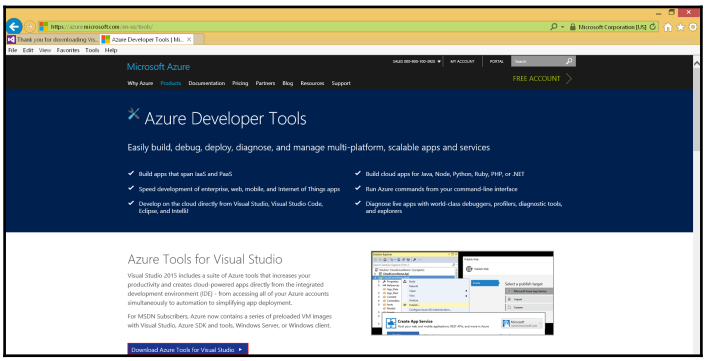

Figure 4: site for Azure Tools for visual studio

This will invoke Web platform installer for installing “Microsoft Azure SDK for .NET(VS 2015)” as shown in figure 5. Click on Install button to install Azure tools. Another prompt to accept the license agreement is shown. Click “Accept” to start the installation of tools. This will also install Visual ARM visual studio template for authoring ARM templates.

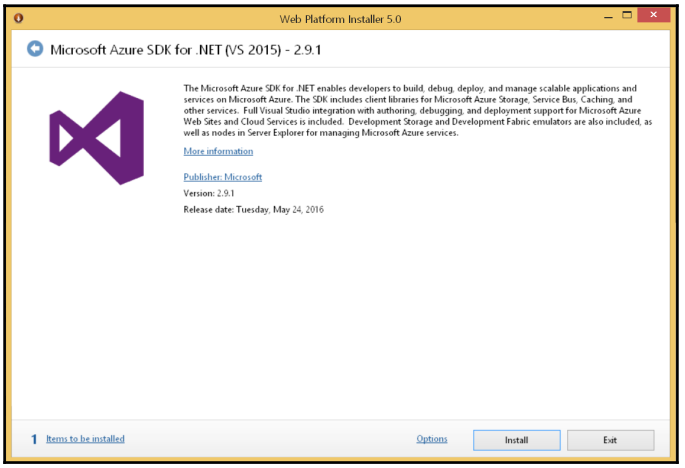

Figure 5: Web Platform Installer

3. Open visual studio and select “new project”, “Azure Resource Group” from cloud category. Name the project as MinimalTemplate, provide C:\templates as

location, the solution name defaults to name of project (you can change solution name to a different name from project name) and click ok as shown in figure 6.

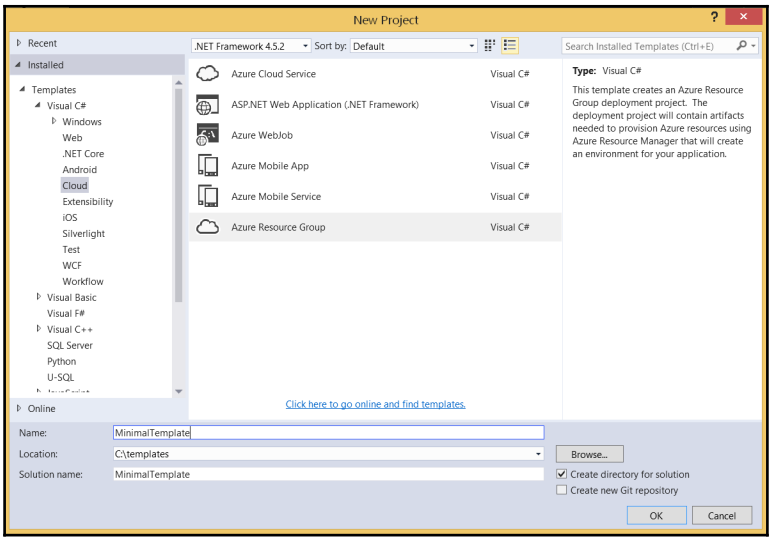

Figure 6: visual studio project creation for Azure Resource Group

4. Select “Blank Template” from list of templates and click ok as shown in figure 7. This is will create a solution and Azure resource group project within it.

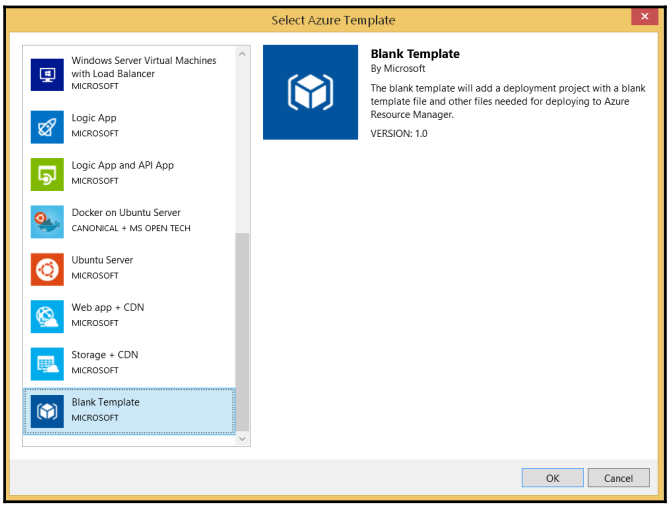

Figure 7: Selecting Blank Template for project

5. The resultant Blank Template is shown here in figure 8.

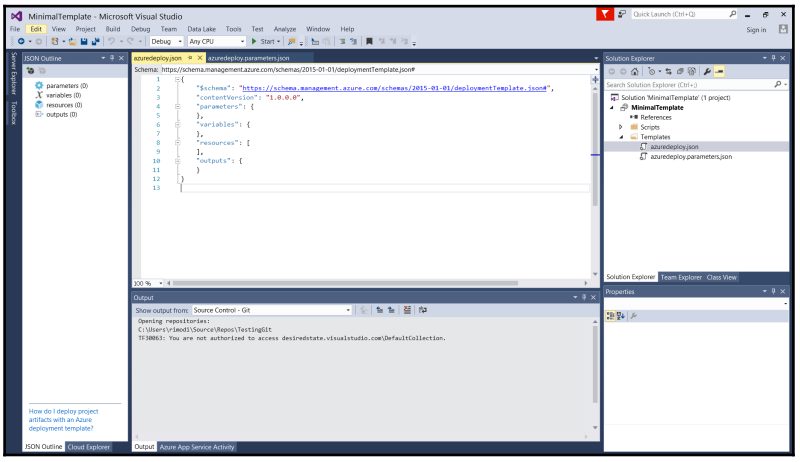

Figure 8: An empty blank Azure Resource Group Template

6. Modify the azureDeploy.json file to reflect the minimal template we created

earlier in this chapter as shown in figure 9 and save the entire project.

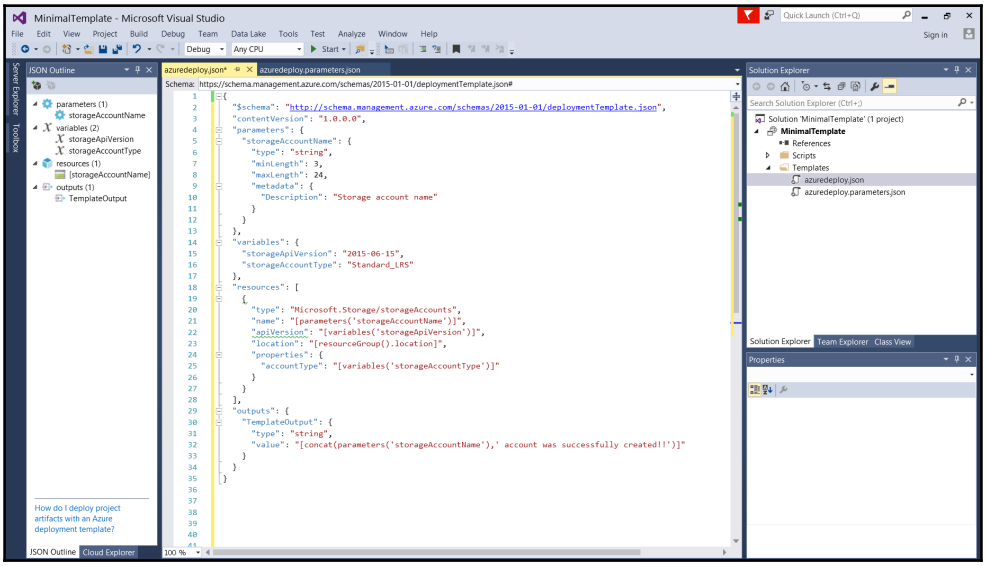

Figure 9: Minimal Template in visual studio

More resources can be added to template by using the “JSON Outline” pane. It has “Add Resource” button that can be used for the same. Later, we will deploy this template using powershell in this chapter.

Deployment tools

Templates can be deployed to a resource group through multiple ways.

- Powershell
- Azure Command line interface
- REST API's

This book uses Powershell and Azure Resource manager module for deploying ARM templates to resource groups. We will look into steps to use Powershell for deployment of templates when we discuss Powershell in details in next section.

Deployments

Powershell allows two modes of deployment of templates

- Incremental
- Complete

Incremental deployment adds resources declared in the template that don't exist in Resource Group, leaves resources unchanged in resource group that are not part of template definition, leaves resources unchanged in resource group that exists in both the template and resource group with same configuration state.

Complete deployment on the other hand, adds resources declared in template to the resource group, deletes resources that do not exist in the template from resource group and leaves resources unchanged that exist in both resource group and template with same configuration state.

Powershell

Powershell is object based command line shell and scripting language used for administration, configuration and management of infrastructure and environments. It is built on top of .net framework and provides automation capabilities. Powershell was released in 2006 as version on Windows. Powershell version 5 is the current version that is available on Windows server 2016 and windows 10 out of box. Powershell 5 is also available as part of windows management framework (WMF) 5.0.

Powershell has truly become a first class citizen among IT administrators and automation developers for managing and controlling the windows environment. Today, almost every windows environment and its components can be managed by Powershell. In fact, almost every aspect of Azure can also be managed by Powershell.

Powershell can be divided into two components.

1. **Powershell Engine:** This is the core engine responsible for executing the Powershell commands executed through the command line, pipelines, scripts and functions. The engine provides the execution environment and context in terms of security, concurrency, threading, base .net framework, extended type system, modules, logging and auditing. Powershell engine exposes runspace interfaces that are used by Powershell hosts for interacting with it.
2. **Powershell Host:** They are user facing application or command line interface responsible for interacting with user, accepting inputs from them and passing

them to Powershell Engine for execution using the runspace interface provide by engine. The return values from engine are displayed to the user using the same host. Each Powershell host has its own configuration that are accessible through host provided system variable "$Host". There are two hosts provided out of box in Windows

- Powershell.exe
- Powershell Integrated scripting environment (ISE)

Powershell ISE provided added functionality for authoring Powershell scripts apart from being a command line interface.

Powershell Features

Powershell provides lots of features and capabilities and demands a complete book by itself. In this book, we will look into some of the important aspects of powershell relevant to us for DevOps.

Cmdlets

Powershell provides small executable commands called "cmdlets" that executes a single task or operation. There are hundreds and thousands of cmdlets available in Powershell. My own machine running windows 8.1 has more than 3000 cmdlets available. There is a cmdlet available for almost every administration and management activity in Powershell. Cmdlets names follows verb-noun naming convention. For example, *Get-Process* cmdlet is responsible for retrieving all running processes in a system. *Get* refers to the verb or action being performed and *Process* refers to the subject or noun under consideration. Powershell accepts object oriented objects as arguments for their parameters and returns back well defined objects as return value.

Pipeline

Powershell is object based language which helps in implementing the concepts of pipelines. Pipelines refers to a series of cmdlets defined in a single statement, executed one after another where the output of previous cmdlet becomes the input of next cmdlet. Pipelines are defined using the pipe character '|'.

An example of pipeline is shown here

```
Get-Process -Name Notepad | stop-process
```

The code shows two cmdlets '*Get-Process*' and '*Stop-Process*' in a pipeline. Execution of the statement executed the first cmdlet first. Get-Process outputs objects of type '*System.Diagnostics.Process*'. If Notepad is running on the system, it will return an object containing details about its process. The same process object is send to Stop-Process cmdlets as input arguments which stops the Notepad process from running.

Variables

Powershell provides support for variables which are temporary storage locations in memory to store values that can be used and changed subsequently. They help in making script granular, flexible and maintainable. Powershell does not mandate to specify data types while defining variables. In such case, powershell implicitly decides the data type based on values assigned to the variable. When variables are defined with data types, the variable can accept only those values that adheres to the rules of data type. The native powershell variable data types are string, char, byte, int, long, bool, decimal, single, double, array and hashtable. Variables in Powershell are defined using the dollar symbol "$".

Examples of variables are

The following code assigns a string value to OSName variable. Note the "$" sign prefixed to OSName.

```
$OSName = "Windows Server 2016"
```

The following code assigns the same string value to OSName variable however, this time data type is provided explicitely.

```
[string] $OSName = "Windows Server 2016"
```

Variables can be referred by their names prefixed with the dollar "$" symbol.

$OSName would print the value stored in OSName variable.

Scripts and Modules

Powershell is also a scripting language. We can use cmdlets, functions, pipelines and variables within scripts as well. Scripts are reusable code that can be executed multiple times. They encapsulate code, function, data and logic in a script file with ps1 extension. The scripts can be loaded and executed in current execution environment using technique called "*dot-sourcing*". Dot-sourcing refers to powershell syntax to load a file using dot notation as shown here. Please note the "." before the script path.

```
. ./SampleScript.ps1
```

Powershell modules are the means to share scripts, functions and cmdlets with others. Modules are packaged in a specific format that can be readily deployed on any windows system. There are well defined locations in windows operating system that houses modules. System module are available at C:\Windows\System32\WindowsPowershell\1.0\Modules folder location while modules from third party sources are available at C:\Program Files\WindowsPowershell\Modules folder location. C:\ drive refers to the system drive here which might be different for you.

Get-Module cmdlet provides information about all currently loaded modules while *Get-Module -ListAvailable* provides information about all modules that are available on the system but not loaded in the workspace.

Azure Powershell Development environment

Powershell will be used heavily in this book across chapters. Since, all our environments would be hosted on Azure, Azure Powershell module will be a core module for managing Azure through powershell. One of the way to deploy ARM templates is through Powershell. Before we can use Powershell to deploy templates in Resource groups, we need to install the Azure ARM powershell modules. These modules providers the cmdlets, functions and code to connect and authenticate to Azure, Create Resource groups and deploy templates in them. They also provide functionality to peek into the Azure Resource groups logs to check the status of deployment and also troubleshoot issues. Azure powershell module is available through Web platform installer as well as through Powershell Gallery. Windows Server 2016 and Windows 10 provides package management and PowershellGet modules for quick and easy download and install of powershell modules from powershell gallery. The cmdlets from these modules are already configured with powershell gallery source and repository information. PowershellGet module provides “Install-Module” cmdlet for downloading and installing modules on system. Installing of module is a simple activity of copying the module files at well-defined module locations.

Before moving ahead in this section, there should be powershell host, either powershell or powershell ISE opened as an administrator. Downloading and installing modules requires administrative privileges.

AzureRM modules enables working with Azure ARM from a client machine using powershell. Let's download and install AzureRM module using Install-Module cmdlet. Before using any cmdlet, we should explicitly import the module in current workspace as a good practice, although powershell will auto load the module if it's not already loaded. This is shown here.

```
Import-module PowershellGet
Install-Module -Name AzureRM -verbose
```

"Install-Module" cmdlet is dependent on Nuget provider to interact with Nuget based repository. The first time a package management cmdlet is used, it will ask to download the same. Click on "Yes" to any prompts related to downloading Nuget provider. This is shown in image 10.

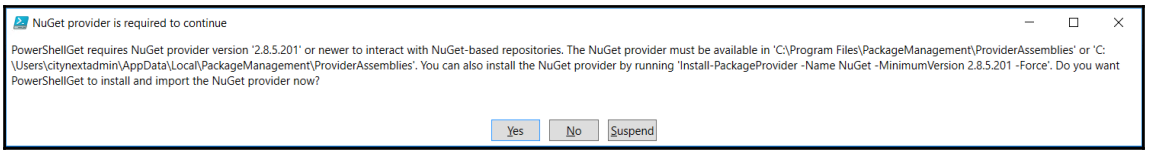

Figure 10: Permission to install Nuget Package Provider

There would be another prompt stating that the module is downloaded from an untrusted repository. "PSGallery" repository is not trusted by default. Accept "yes" to download the provider. This is shown in figure x.

Figure 11: Permission to download from untrusted repository

These prompts can be disabled by using the force switch as show here

```
Install-Module -Name AzureRM -Verbose -Force
```

AzureRM module is a container module that references individual sub-modules each representing a resource provider namespace. Azure ARM functionality is divided into multiple modules, each module representing a ARM resource provider and they contain provider specific cmdlets and functionality. It is responsible for installing all ARM module. AzureRM module can be used to download all sub-modules in one go and it can also be used to download individual sub-modules. In precedent code, the module of module to download is "AzureRM". This will download all sub-module. Similarly, to download only a single or few modules, the code shown next can be used. To import individual modules, explicit name should be provided to this cmdlet. The first line downloads a single module "AzureRM.Compute" and second one install two sub-modules: "AzureRM.Storage" and

"AzureRM.Network".

```
Install-Module -Name AzureRM.Compute -Verbose
Install-Module -Name AzureRM.Storage, AzureRM.Network -Verbose
```

Before we can do anything with resource groups and templates, we should authenticate with Azure. This is done through "Login-AzureRMAccount" cmdlet as shown here.

```
Login-AzureRmAccount
```

It is to be noted that cookies must be allowed on the computer from where the authentication is initiated.

This cmdlet can be configured to use username and password defined in code or it will open a login window. This is shown in Figure 12. Please enter valid username and password to authenticate and login to Azure.

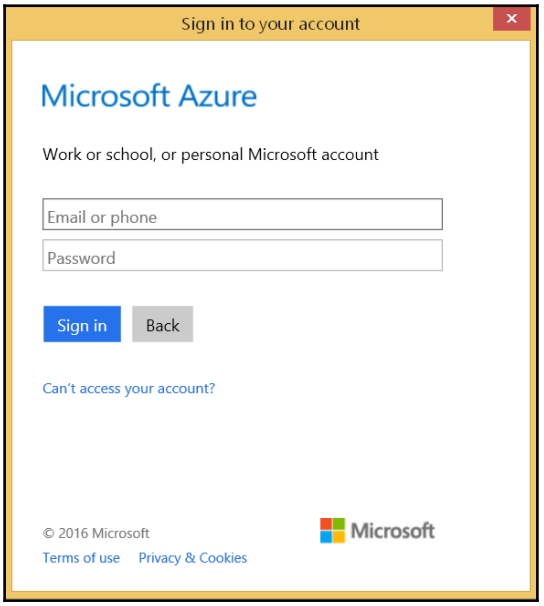

Figure 12: Azure Login window using Powershell

As seen before, templates are deployed within a Resource group. We should be able to create a resource group using powershell and then deploy a template within it after successful login to Azure. This is shown here

```
Import-module AzureRM.Resources
New-AzureRmResourceGroup -Name "myResourceGroup" -location "West Europe" -verbose
```

Where name refers to the name of newly created resource group and location refers to a valid Azure region for resource group to be created. Verbose provides additional information while executing the cmdlet.

After resource group is created, templates can be deployed into it.

To deploy a template, cmdlet “New-AzueRmResourceGroupDeployment” can be used and this is shown here

```
New-AzureRmResourceGroupDeployment -Name "FirstDeployment" -ResourceGroup
"myResourceGroup" -TemplateFile "C:\template\azureDeploy.json" -verbose
```

Here name refers to the name of deployment which is any valid name that can be used to identify the deployment, resource group refers to the name of resource group where this deployment will provision resources and templateFile refers to file location of ARM template.

To deploy the Minimal Template that was created earlier, execute the following command from any powershell editor. It is to be noted that storage name must be unique across azure and resource group must be unique within a subscription. The resource group is named “azureRG” provisioned at West Europe region. The minimal template takes “storageAccountName” as a parameter. This parameter must be supplied when deploying the template as shown here.

```
New-AzureRmResourceGroup -Name "azureRG" -Location "West Europe" -Force -Verbose
New-AzureRmResourceGroupDeployment -Name "Deploy1" -ResourceGroupName "azureRG"
-Mode Incremental -TemplateUri
C:\templates\MinimalTemplate\MinimalTemplate\Templates\azuredeploy.json" -Verbose -
storageAccountName "auniquename"
```

Pester

Now, it's time to visit another powershell utility called Pester. Pester helps in defining and executing tests on powershell scripts, functions and modules. These could be unit tests, functional tests or operational validation tests. Pester is an open source utility freely downloadable from GitHub as a powershell module. It is also available out of box on Windows 10. We will use pester to define unit tests and operational validation tests and also execute them in continuous integration build pipeline and continuous deployment release pipeline in this book. As of writing, Pester 3.0 is the latest stable version available for download, is used in this book.

Pester helps in defining unit tests in a simple English style language using simple constructs like “Describe” and “It” and validates them through assertions. Pester can be downloaded as zip file from https://github.com/pester/pester/archive/master.zip. After downloading, the zip file should be unblocked and its content should be extracted to a well-defined module location. We already know that modules live at both $env:windir\system32\WindowsPowershell\v1.0\modules folder and $env:ProgramFiles\WindowsPowershell\modules folder and pester should be extracted to any one of these folder locations. We will use modules folder from $env:ProgramFiles to store Pester module.

Install Pester

The steps mentioned in previous section will be automated through powershell for installing pester. The entire script Install-Pester.ps1 is shown here.

```
Param(
   # folder location for storing the temp pester downloaded files
   [string]$tempDownloadPath
)
# create a temp folder for downloading pester.zip
New-Item -ItemType Directory -Path "$tempDownloadPath" -Force

# download pester.zip from GitHub
Invoke-WebRequest -Uri https://github.com/pester/pester/archive/master.zip -OutFile
"$tempDownloadPath\pester.zip"

# files from internet are generally blocked. unblocking the archive file
Unblock-File -Path "$tempDownloadPath\pester.zip"

# extracting files from archive file to Well-defined modules folders
Expand-Archive -Path "$tempDownloadPath\pester.zip" -DestinationPath
$env:ProgramFiles\WindowsPowershell\Modules" -Force

# renaming the folder from pester-master to pester
Rename-Item -Path "$env:ProgramFiles\WindowsPowershell\Modules\Pester-master"
NewName "$env:ProgramFiles\WindowsPowershell\Modules\Pester" -ErrorAction Continue

# test to check if pester module is available
Get-Module -ListAvailable -name pester
```

Use the code shown here to execute the script and installing pester on a system. This code assumes that Install-Pester.ps1 script is available at C:\

```
C:\Install-Pester.ps1 -tempDownloadPath "C:\Pester"
```

Let me explain each line within the script.

The script starts with accepting a single parameter "$tempDownloadPath" of type string. This path should be provided by the caller as argument to the script.

```
Param (
   # folder location for storing the temp pester downloaded files
   [string]$tempDownloadPath
)
```

The next statement creates a new folder at the location provided by user. Force ensures that an error is not generated even if a folder with same name exists.

```
New-Item -ItemType Directory -Path "$tempDownloadPath" -Force
```

Then, Invoke-WebRequest cmdlet is used to download the pester archive file from GitHub and store the downloaded file as pester.zip at user provided folder.

```
Invoke-WebRequest -Uri https://github.com/pester/pester/archive/master.zip -OutFile
"$tempDownloadPath\pester.zip"
```

Next, the downloaded file is un-block for further use using unBlock-File cmdlet. Blocked files cannot be opened.

```
Unblock-File -Path "$tempDownloadPath\pester.zip"
```

Now, the archive file is expanded, all its files and folders are extracted to modules folder which is a well-defined module location

```
Expand-Archive -Path "$tempDownloadPath\pester.zip" -DestinationPath
"$env:ProgramFiles\WindowsPowershell\Modules" -Force
```

The extracted folder name is Pester-master however; the name should be Pester. Rename-item cmdlet is used to rename pester-master to pester. This cmdlet throws an error if folder named Pester-master does not exists. Normally, the script will terminate if an error occurs. With ErrorAction as continue, rename-item will throw an error but script execution will not stop.

```
Rename-Item -Path "$env:ProgramFiles\WindowsPowershell\Modules\Pester-master" -
NewName "$env:ProgramFiles\WindowsPowershell\Modules\Pester" -ErrorAction Continue
```

Finally, a small test is conducted to ensure Pester module is available using Get-Module cmdlet. If this cmdlet outputs the name and version of pester module would mean that Pester is successfully installed on the machine.

```
Get-Module -ListAvailable -name pester
```

Writing tests with Pester

Pester provides an easy to use “New-Fixture” cmdlet. Executing this cmdlet creates two files. The first file is for authoring powershell scripts and functions and second file is for authoring unit tests for scripts and function in the first file. This cmdlet also ties both the files together by generating scaffolding powershell code in such a way that the entire script in first file in loaded into the workspace of second file when the second file containing unit tests are executed. The scaffolding code ensures that powershell scripts and functions are available to the test cases for validation. Pester should be able to execute any code from first file that are referred in test cases. It is important to understand that “New-Fixture” cmdlet helps in importing a script into another workspace. This can be done manually by dot-sourcing the script files into pester unit tests file. In fact, New-fixture cmdlet also dot-sources the script file into unit tests file.

Understanding pester is much easier by experiencing it rather than going through its theory. Let's understand Pester through a scenario of writing a simple function for adding two numbers and corresponding tests for testing the same through individual steps.

1. Let's create a folder at C:\ that will house both our scripts and pester unit tests. Let's name it “addition”.
2. From any powershell host execute the commands shown here to generate the script files. The cd command changes the directory to C drive. Import-Module cmdlet imports the Pester module into current workspace and New-Fixture cmdlet from Pester module creates the script files at addition folder location. It will generate two files – Add-Numbers.ps1 and Add-Numbers.Tests.ps1

   ```
   cd C:
   Import-Module -Name Pester
   New-Fixture -Path "C:\Addition" -Name "Add-Numbers"
   ```

   The script generated in Add-Numbers.ps1 is an empty Add-Numbers function shown here.

   ```
   function Add-Numbers {
   }
   ```

   The script generated in Add-Numbers.Tests.ps1 is shown here. The first three statements get the path of the add-Numbers.ps1 script file and loads it in current runspace through dot-sourcing.

   Variable $here contains the parent folder of Add-Numbers.Tests.ps1 i.e. “C:\addition”.

The second statement gets the file name of tests script file i.e. Add-Numbers.Tests.ps1 and replaces the \.Tests\. with a single '.' And assigns to the variable $sut. $sut contains the value Add-Numbers.ps1 which is actually our script file containing application logic.

The third statement simply combines both the folder path and script file name and load it by dot-sourcing it. This makes out Add-Numbers function available to the test cases defined in the tests script file.

```
$here = Split-Path -Parent $MyInvocation.MyCommand.Path
$sut = (Split-Path -Leaf $MyInvocation.MyCommand.Path) -
replace '\.Tests\.', '.'
. "$here\$sut"
Describe "Add-Numbers" {
   It "does something useful" {
      $true | Should Be $false
   }
}
```

The last three statements refer to a single test case generated by New-Fixture cmdlet.

“Describe” refers to a collection of tests cases. It is a container that can contain multiple tests. It is actually a function defined in Pester module that accepts a script block. It should be named to reflect the which component is getting tested.

“it” refers to a single test case and its naming should indicate the nature of test performed. It is also a function defined in Pester module that accepts a script block. This script block contains the actual tests and assertions. The assertions determine whether the test is successful or not. A successful assertion is shown in green colour while failures are shown in red colour. “Should be” is an assertion command. There are multiple assertions provided by pester like “Should be”, “Should BeExactly”, “Should match”, “Should Throw” and more.

3. Modify the Add-Number.ps1 script file with real code. The Add-Number script look like this

```
function Add-Numbers {
   param (
      [int] $Num1,
      [int] $Num2
   )
   return $Num1 + $Num2
}
```

The Add-Numbers function has been modified to accept two parameters, Num1 and Num2 both of datatype integer. It adds both the numbers and return back the same to the caller.

4. The Add-Numbers.tests.ps1 script has been modified by removing the default test provide by Pester and adding two test cases within the same "describe" section. "Describe" section has been renamed as "test cases adding two numbers"

   The first test named "checking when both the numbers are positive" declares two variables $firstNumber and $secondnumber and assign values to them. It invokes the Add-Number function passing both the variables as arguments to it. The return value from the function is piped and asserted (verified). Similarly, the second test named "checking when One number is positive and another negative" again declares two variables $firstNumber and $secondnumber and assign values to them. However, this time the value of one of the variable is negative. It invokes the Add-Number function passing both the variables as arguments to it. The return value from the function is piped and asserted.

```
$here = Split-Path -Parent $MyInvocation.MyCommand.Path
$sut = (Split-Path -Leaf $MyInvocation.MyCommand.Path) -replace
'\.Tests\.', '.'
. "$here\$sut"
Describe "test cases adding two numbers" {
   it "checking when both the number are positive" {
      $FirstNumber = 10
      $SecondNumber = 20
      Add-Numbers -Num1 $FirstNumber -Num2 $SecondNumber |
      should be 30
   }
   it "checking when One number is positive and another
   negative" {
      $FirstNumber = -10
      $SecondNumber = 20
      Add-Numbers -Num1 $FirstNumber -Num2 $SecondNumber |
      should be 10
   }
}
```

5. Its time now to run the tests. The test can be executed by using "Invoke-Pester" cmdlet provided by Pester module. This cmdlet takes the path of the Test scripts file. Execute the cmdlet as shown here to execute the tests written earlier.

```
Invoke-Pester -Script "C:\Addition\Add-Numbers.Tests.ps1"
```

   This will invoke the scripts and execute all the test cases described in it. The same

is shown in Figure 13. The Green colour denotes successful test pass and red one means failed test case.

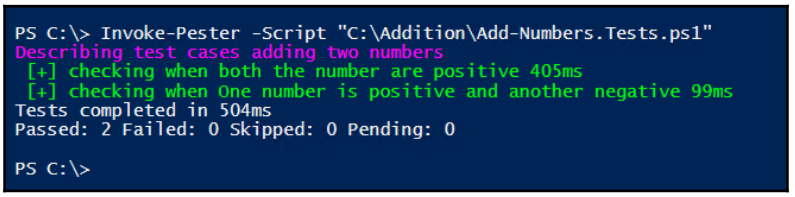

Figure 13: Executing Pester unit tests

Pester real time example

Let's work on another example on Pester. This time we will write test for ensuring whether a website and its related application are in working condition on a web server. This time New-Fixture is not used for generating the script files. Instead, both the application code and test cases are written from scratch using a powershell ISE editor.

Both the code for provisioning of web server artifacts and related tests are within the same file although they can be in different files as seen before.

The entire code is shown here. The script is stored at C:\Test-WebServer.ps1. A "CreateWebSite" function is defined taking four parameters. These parameters capture inputs for Application pool name ($appPoolName), Website name ($websiteName), its port number ($port) and the path referred by the website ($websitePath). The function first creates an IIS application pool first using New-WebAppPool cmdlet and $appPoolName parameter and then creates a IIS website using New-WebSite cmdlet using all the four parameters.

```
function CreateWebSite
{
   param(
      [string] $appPoolName,
      [string] $websiteName,
      [uint32] $port,
      [string] $websitePath
   )
   New-WebAppPool -Name $appPoolName
   New-Website -Name $websiteName -Port $port -PhysicalPath $websitePath -
   ApplicationPool $appPoolName -Force
```

```
}
Describe "Status of web server" {
   BeforeAll {
      CreateWebSite -appPoolName "TestAppPool" -websiteName "TestWebSite" -
      port 9999 -websitePath "C:\InetPub\Wwwroot"
   }
   AfterAll {
      Remove-Website -Name "TestWebSite"
      Remove-WebAppPool -Name TestAppPool
   }
   context "is Website already exists with valid values" {
      it "checking whether the website exists" {
         (Get-Website -name "TestWebSite").Name | should be "TestWebSite"
      }
      it "checking if website is in running condition" {
         (Get-Website -name "TestWebSite").State | should be "Started"
      }
   }
}
```

The test cases are written in the same file. “Describe” named “Status of web server” starts the test cases. A special construct “BeforeAll” and “AfterAll” is used within the “Describe” block. “BeforeAll” runs the script within it just once for all test cases in a “Describe” block before the execution of the first “It” block. Similarly, the script within “AfterAll” is executed after all the test cases (“it” blocks) has been executed. They are typically used for setting up and cleaning up the environment. Here, we invoke our function “CreateWebSite” within “BeforeAll” block to provision our application pool and web site. Both the application pool and website is removed in “AfterAll” block. This ensures that the environment is in same state as before the start of the tests.

“Context” is also a new container construct that can contain and group multiple test cases (“It” blocks). Context does not affect the execution of tests. They remain the same as before however it adds addition metadata and groups tests based on condition. For example, a context is group of test cases with different valid values while another context is group of test cases with invalid values.

Two “It” test cases are show. The test case “checking whether the website exists” uses the Get-WebSite cmdlet to get the name of the website and assert if it is same as that the function created. The test case “checking if website is in running condition” again uses the same cmdlet but checks its status property and compares with “Started” value for assertion.

Executing the above script with Invoke-Pester cmdlet shows the result as shown in Figure 14.

```
Invoke-Pester -Script "C:\Test-WebServer.ps1"
```

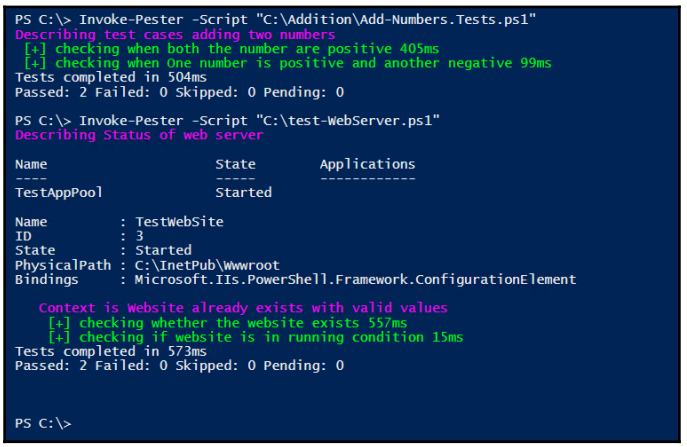

Figure 14: Executing Pester tests

It is important to note the naming pattern of “Describe”,” Context” and “it” blocks. They have been named in a way such that the result of executing the unit tests can be read as simple English which is meaningful and provides enough context about tests that are successful and ones that failed.

Desired State Configuration

Desired State configuration(DSC) is a new configuration management platform from Microsoft build as an extension to powershell. DSC was originally launched as part of WMF 4.0. It is available as part of Windows Management Framework(WMF) 4.0 and 5.0 for all windows server operating system above Windows 2008 R2. WMF 5.0 is available out of box on Windows Server 2016 and Windows 10. It uses the core infrastructure of Web services for Management (WSMAN) and Windows Remote Management (WinRM) for its working. It is an extension to powershell and adds language constructs, features and cmdlets for easy authoring and execution of configuration across heterogeneous environments.

DSC is declarative language enabling Infrastructure as code by representing and describing the entire infrastructure and its configuration through code. DSC configuration files are simple .ps1 script files that can be stored in source control repositories for version control.

DSC represents the target state and configuration for environments through code. It represents “what” the environment state and configuration should look like. The “How” part is not required which is taken care by DSC internally.

DSC will be used in Release pipeline to configure multiple environments and application configuration.

DSC Push architecture

A simple DSC Configuration is shown here.

```
Configuration EnableWebServer
{
   Import-DscResource -ModuleName 'PSDesiredStateConfiguration'
   Node WebServer01
   {
      WindowsFeature IIS
      {
         Name = "Web-Server"
         Ensure = "Present"
      }
   }
}
EnableWebServer -OutputPath "C:\DSC-WebServer" -Verbose
Start-DscConfiguration -Path "C:\DSC-WebServer" -Wait -Force -Verbose
```

The code declares “EnableWebServer” configuration. This configuration is responsible for ensuring the presence of Web-Server (IIS) windows feature on “WebServer01” computer. The node name should be changed reflecting an actual computer name in your network. Notice that the configuration does not use any scripting or programming to provision a web server. The configuration is merely providing the intent and its configuration values, the rest is taken care by DSC resources internally. The node section comprises of names of computers that this configuration is applicable to. WindowsFeature mentioned in configuration is a DSC resource that actually provides the logic for provisioning of windows feature on target computer. The “How” part is taken care by these DSC resources.

Executing the above configuration as shown in the code, generates the configuration Management Object Format(MOF) file for each node name in configuration at folder location provided to the path parameter. When the path parameter is omitted, the MOF files will be generated at current working folder. These MOF files are pushed to target nodes by DSC using the “Start-DSCConfiguration” cmdlet. This cmdlet takes path parameter as input representing the folder containing MOF files, loads all MOF files and send them to their respective computer nodes using the MOF file names.

Alternatively, you can also provide ComputerName parameter to Start-DSCConfiguration cmdlet which will load MOF file matching the ComputerName and push it to that node only. This is shown here.

```
Start-DscConfiguration -Path "C:\DSC-WebServer" -ComputerName WebServer01 -Wait -Force
-Verbose
```

Installing WMF 4.0 or 5.0 on client nodes ensures the installation of DSC Local Configuration Manager(LCM). It is responsibility of LCM on these client nodes to accept the configuration sent to it and execute them on the local machine. LCM treats the received configuration as Golden configuration. LCM is configured to run periodically, checking the current state of configuration with Golden desired state of configuration. If it finds any deviation, it brings back the configuration and environment back to desired state. This ensures that target servers and their configuration can be auto-remediated in case there are changes applied to them.

DSC resources must be available at the client nodes for LCM to periodically check and auto-remediate the configuration. They help LCM in validating the current resource configuration with desired configuration, bringing back the current configuration to desired configuration. Earlier we witness a single DSC resource in configuration. There are more than three hundred DSC resources available to be used within configurations. While DSC comes with few out of box DSC resources, it is possible to author custom DSC resources. DSC provided resources are WindowsProcess, WindowsFeature, Service, File, Archive, User, group, package, Log and script.

The DSC Push architecture is shown in Figure 15.

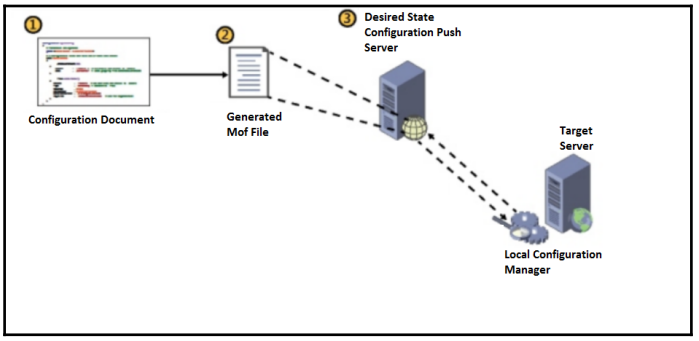

Figure 15: Desired State Configuration Push architecture

DSC Pull architecture

Whatever, we saw till now was one of the DSC architecture implementation called the Push Mode. DSC comes in two architectural implementation modes.

- DSC Push Mode and
- DSC Pull Mode.

The example shown before depicted DSC Push mode architecture. In Pull mode, the configurations are not pushed to client nodes. Instead, LCM on client nodes are configured with appropriate endpoint information through which it is able to connect and download configuration from DSC Pull Servers and execute them on local server. It is needless to say that DSC Pull Servers needs to be created and available with appropriate published configuration files before they are fetched by LCM.

DSC Pull mode is decentralized and scalable way of enabling configuration management. Hundreds and thousands of client nodes can simultaneously pull configurations automatically without any manual intervention. DSC resources can also be downloaded along with configuration files eliminating the need to pre-install resources on client nodes before executing the configuration. It is much more manageable and flexible compared to DSC Push mode. We will be using DSC Pull Server and configuration to create our base container images.

The DSC Pull architecture is shown in Figure 16.

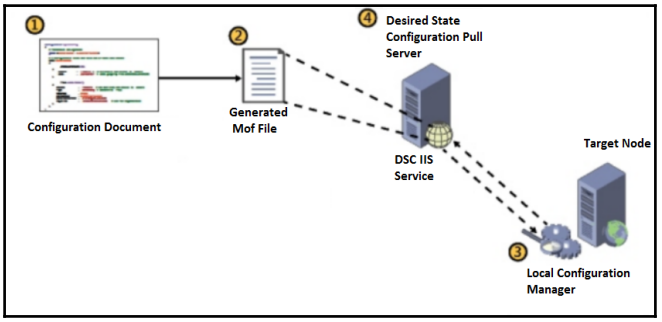

Figure 16: Desired State Configuration Pull architecture

Figure 6 shows the DSC configuration is authored, DSC MOF file is generated and published on a Pull Server. The LCM of client node downloads the configuration and reconfigures the node according to the configuration.

Figure 7 shows the DSC Pull mode request flow between client node and pull server.

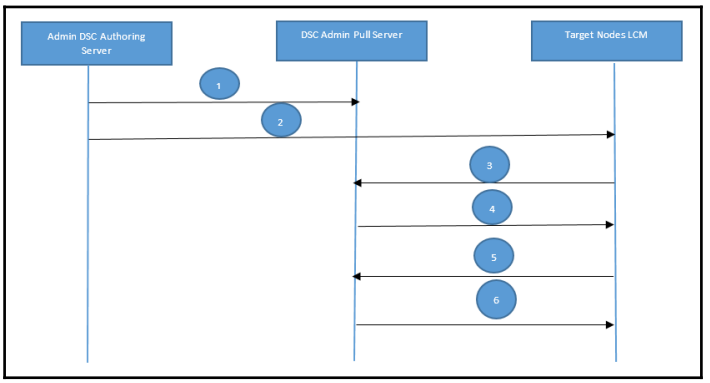

Figure 17: Desired State Configuration Push flow sequence

The following Process is used during Pull Mode

1. The Configuration developer creates the configuration, generates MOF files and other artifacts and copies them to pre-defined folder location on to the DSC Pull Server. This is simply a copy paste operation.
2. DSC administrator configures LCM of every client node's with information about pull server endpoint, configuration details and frequency to pull configuration.
3. The client node sends request to pull server for fetching configuration.
4. Pull server on receiving the request responds by sending configuration information.
5. Steps 2 to 4 about is repeated based on configured frequency.
6. For every request received, response is sent with configuration information to the client nodes.

Pull Configuration example

The configuration shown next is used to explain DSC Push architecture

```
Configuration EnableWebServer
{
   Import-DscResource -ModuleName 'PSDesiredStateConfiguration'
   Node EnableWebServer
   {
      WindowsFeature IIS
      {
```

```
            Name = "Web-Server"
            Ensure = "Present"
        }
    }
}
```

You will notice that name of Node is no more a computer node name. This is because any node can pull this configuration from pull server. The node name is of high significance in DSC. The Node name decides the name of the generated MOF file. In case of pull architecture, even the name of configuration has impact on the way these configuration is downloaded by target node. The name used reflects the intent of the configuration.

You will also notice that start-DSCConfiguration cmdlet is not used for this configuration. Start-DSCConfiguration is used to push configuration to target nodes. For pull server configurations, there are two additional requirements.

1. Instead of pushing, configurations are deployed to a well-known folder path known to pull server. Pull server can be an IIS website or a SMB share. In this book, IIS websites will be configured as pull server. Pull server is configured with this folder path in its web.config file. When a request for configuration arrives at pull server, the website accesses this folder path, loads and sends back the configuration. By default, this folder location is "$env:ProgramFiles\WindowsPowershell\DSCService\Configuration". We generate MOF files as we did in Push scenario by executing the configuration with output path as "$env:ProgramFiles\WindowsPowershell\DSCService\Configuration". This would generate the MOF file within output path folder. This is shown here.

   ```
   EnableWebServer -OutputPath
   "$env:ProgramFiles\WindowsPowershell\DSCService\Configuration" -
   Verbose
   ```

2. A checksum/hash should be generated for each MOF file. This hash is saved in .mof.checksum file along with the original mof file. Each .mof file should have a corresponding .mof.checksum file with same name. This checksum is important for LCM to validate and decide if it needs to download the configuration from pull server. Every time a LCM on a node pulls a configuration, it sends the hash it already has to pull server. If the hash is different on pull server, configuration is downloaded otherwise it would mean the configuration on node is same as that on pull server. To generate the checksum and write it to a file, DSC provides a cmdlet "New-DSCChecksum". Its usage is shown here. It takes configuratioPath as path to configuration MOF file and Outpath as folder location to generate the corresponding checksum file.

```
New-DSCCheckSum -ConfigurationPath
"$env:ProgramFiles\WindowsPowershell\DSCService\Configuration\
EnableWebServer.mof" -OutPath
"$env:ProgramFiles\WindowsPowershell\DSCService\Configuration" -
Force
```

It is to be noted that if ConfiguationPath parameter refers to a folder rather a MOF file, checksum file will be generated for all MOF files at location referred by OutPath parameter. This is shown here.

```
New-DSCCheckSum -ConfigurationPath
"$env:ProgramFiles\WindowsPowershell\DSCService\Configuration" -
OutPath
"$env:ProgramFiles\WindowsPowershell\DSCService\Configuration" -
Force
```

The configuration can now be pulled by any authorized and configured computer node. After storing configurations on pull server, LCM of computer nodes should be configured so that they can pull configurations from pull server.

To configure LCM on a target node, DSC configuration should be created containing resources specific to LCM. MOF file should be generated and pushed to target nodes, executed locally to LCM to re-configure itself. The approach to push LCM configuration to target node is same as that of any other configuration using Start-DSCConfiguration cmdlet. However, there are differences in the way the configurations are authored. This is explained next.

LCM configuration have “DSCLocalConfigurationManager()” attribute at configuration level. This attribute ensures and enforces that only configuration related to LCM are allowed within the configuration. There are three resources used in this example: Settings, ConfigurationRepositoryWeb and PartialConfiguration.

Settings resource configures the LCM with multiple properties. If you want to view LCM properties, use Get-DSCLocalConfigurationManager cmdlet to list all properties along with their values. The properties used in this example are explanation here.

Property Name	**Description**
ConfigurationModeFrequencyMins	Represents the frequency (in minutes) at which the LCM attempts to executes and applies the current configuration on the target node. The default value is 30. It should be set to an integer multiple of RefreshFrequencyMins.

RefreshMode	Possible values are **Push** (the default), **Disabled** and **Pull**. In the "push" configuration, the configuration must be pushed to each target node. In the "pull" mode, a "pull" server should be available hosting configurations, for Local Configuration Manager to contact, access, download and apply the configurations.
RefreshFrequencyMins	Represents the frequency (in minutes) at which the Local Configuration Manager contacts the "pull" server to download the current configuration. This value can be set in conjunction with ConfigurationModeFrequencyMins. When RefreshMode is set to PULL, the target node contacts the "pull" server at an interval set by RefreshFrequencyMins and downloads the current configuration. At the interval set by ConfigurationModeFrequencyMins, LCM applies the latest configuration that was downloaded onto the computer node. The default value is 15.
ConfigurationMode	This property has three possible values. This property determine how LCM should behave in the event of configuration drifts or availability of newer configuration. It can take the following values: • **ApplyAndAutoCorrect**: LCM keeps executing the configuration on a regular basis (specified by ConfigurationModeFrequencyMins) without checking whether the configuration is different or not. • **ApplyAndMonitor**: In this mode, which is the default, LCM compares the configuration specified on the "pull" server with the configuration file on the target node. If a difference is detected, the discrepancy is reported in logs, but does not apply the new configuration. • **ApplyOnly**: In this mode, LCM does not automatically run in the background. If a "pull" server is configured, it will check with the server periodically and only if a new configuration is present, it will apply that configuration to the target node.

RebootNodeIfNeeded	Certain configuration changes on a target node might require it to be restarted. Computer node will restart if value is **True**. If value of this property if **False** (the default value), the configuration will complete, but the node must be restart automatically. It should be restarted manually.

ConfigurationRepositoryWeb defines details about Web based pull server. The three property used in this example provides enough information to connect to pull server. CertificateID and AllowUnsecureConnection are other available properties but not used in this example. CertificateID refers to certificate thumbprint for authenticating to pull server and AllowUnsecureConnection accepts a boolean value determining whether unauthenticated access to pull server is allowed.

Property Name	**Description**
ServerURL	The url of Web based pull server. A dummy url is provided in code shown next and should be changed to a valid pull server url.
RegistrationKey	Common key used by both pull server and target node. The key is defined on pull server and only IT administrator should have knowledge about it. The key has been masked in code shown next. Reader should provide their registration key
ConfigurationNames	The name of configuration files available on pull server at well-defined folder accessible by pull server website. The name of configuration available on pull server is WebFeatures

PartialConfiguration section defines the Configuration source that should be used to deploy the configuration on computer node. Partial Configurations help in downloading multiple configuration from multiple pull servers and apply all of them together on local node. In this example, we have used a single configuration in ConfigurationRepositoryWeb and a single PartialConfiguration section, but more configurations can be added to ConfigurationNames property of ConfigurationRepositoryWeb and multiple partial configuration sections can be defined corresponding to those configurations. It is important to note that the name of partial configuration must available in “ConfigurationNames” property of ConfigurationRepositoryWeb. Moreover, the name must also match to the name of the original configuration based on which MOF file is generated. We mentioned at the beginning of this section that configuration names are quite important in pull scenario's.

In short, the Partial configuration name should match the name of original configuration,

the name of MOF file and the name should also be available as part of ConfigurationNames property of ConfigurationRepositoryWeb. The RefreshMode property defined in settings section can be overridden in this section with a different value. If these names do not match, LCM on computer nodes will not be able to pull configuration information from pull servers.

Finally, the configuration is pushed to the localhost computer node (in this case the configuration is authored on local computer) and executed to effect changes to LCM configuration using "Set-DSCLocalConfigurationManager" cmdlet. This cmdlet is responsible for pushing and updating only Local Configuration manager configuration. It cannot be used for pushing other types of configuration.

The configuration assumes that the LCM configuration is authored on the node that would act as a client to pull server. This is the reason the node name is "localhost" in code shown next. However, if you want the below configuration to be authored on any other server, it is possible to do so. In that case, the node name in code should change to actual name of the node and start-DSCConfiguration should either not use the computer name altogether or should provide the actual node name to ComputerName parameter as argument.

Pull server information like RegistrationKey, endpoint url, Configuration Names should be gathered before the below script is executed. The RegistrationKey is available from "RegistrationKeys.txt" file available at "$env:ProgramFiles\WindowsPowerShell\DscService" folder on pull server. The endpoint url of pull server can be obtained from Internet Information Server (IIS).

This chapter assumes that you already have a working pull server in your network. In case, you do not have a pull server available, complete steps for creating a pull server is provided in appendix 1 in this book which would create a pull server with endpoint url "https://10.4.0.4:9100/PSDSCPullServer.svc/" on port 9100.

```
[DSCLocalConfigurationManager()]
configuration PartialConfigurationDemo
{
   Node localhost
   {
      Settings
      {
         ConfigurationModeFrequencyMins = 30
         RefreshMode = 'Pull'
         RefreshFrequencyMins = 30
         ConfigurationMode = "ApplyandAutoCorrect"
         RebootNodeIfNeeded = $true
      }
      ConfigurationRepositoryWeb IISConfig
```

```
        {
            ServerURL = "https://10.4.0.4:9100/PSDSCPullServer.svc/"
            RegistrationKey = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
            ConfigurationNames = @("EnableWebServer")
        }
        PartialConfiguration EnableWebServer
        {
            Description = 'Configuration for installing Web server'
            ConfigurationSource = '[ConfigurationRepositoryWeb]IISConfig'
            RefreshMode = 'Pull'
        }
    }
}
PartialConfigurationDemo -OutputPath "C:\LCMConfiguration" -Verbose
Set-DscLocalConfigurationManager -path "C:\LCMConfiguration" -force -verbose
Update-DscConfiguration -Wait -Verbose
Start-DscConfiguration -UseExisting -Wait -Force -Verbose
```

After the local configuration management settings are configured, it's time to connect to pull server, download and execute configuration on target node. The configuration is pulled using “Update-DscConfiguration” cmdlet. It does not accept any parameters and runs as a job by default. To execute this cmdlet in synchronous mode -wait switch can be used as shown before. Verbose switch provides additional execution steps when this cmdlet is executed.

After pulling configuration the configuration is executed and applied using “Start-DscConfiguration” cmdlet. It is asked to use existing configuration through “useExisting” switch, wait switch runs the cmdlet in synchronous mode, force switch pushes configuration to target nodes even when the LCM on target node is configured with pull configuration and verbose switch provides additional information about the execution.

If there are no errors while executing the code shown here, Internet information server will be installed, available and configured on target node.

Summary

This was a technology heavy chapter and we covered a lot of different technologies. We started with Azure Resource Manager and its features and concepts, moved on to Azure Resource Manager templates, describes various components of templates and how to author them, constructed a simple minimal template, tools for authoring template and process of deploying them through powershell. Then, we discussed Powershell as a command line and scripting language that helps in automation and administration of infrastructure and environment. We also discussed some of its most important concepts like

variables, pipelines, cmdlets, script and module. We covered Powershell script unit testing tool Pester. We tried to understand pester and its concepts using two examples. Finally, we discussed Desired state configuration as a configuration platform with its pull and push architecture, their process flow and ways to use configuration file to configure target computer nodes.