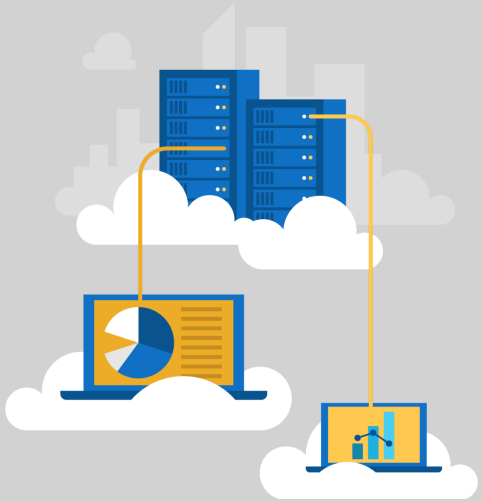
Microsoft

AZ-400.3
Module 1: Design a Release Strategy

Learning Objectives

- Differentiate between a release and a deployment
- Define the components of a release pipeline
- Explain things to consider when designing your release strategy
- Classify a release versus a release process, and outline how to control the quality of both
- Describe the principle of release gates and how to deal with release notes and documentation
- Explain deployment patterns, both in the traditional sense and in the modern sense
- Choose a release management tool

Lesson 01: Introduction to Continuous Delivery



What is Continuous Delivery?

Continuous Delivery (CD) is a set of processes, tools and techniques for the rapid, reliable and continuous development and delivery of software which follows 8 principles:

1. The process for releasing/deploying software must be repeatable and reliable.
2. Automate everything!
3. If something is difficult or painful, do it more often.
4. Keep everything in source control.
5. Done means “released.”
6. Build quality in!
7. Everybody has responsibility for the release process.
8. Improve continuously.

Traditional IT Company

- A fair amount of hotfixes and change requests for production
- Many scope changes during a project
- Lots of unplanned work due to technical debt (environment drift, poor quality, handoffs)
- Business involved but not attached to IT

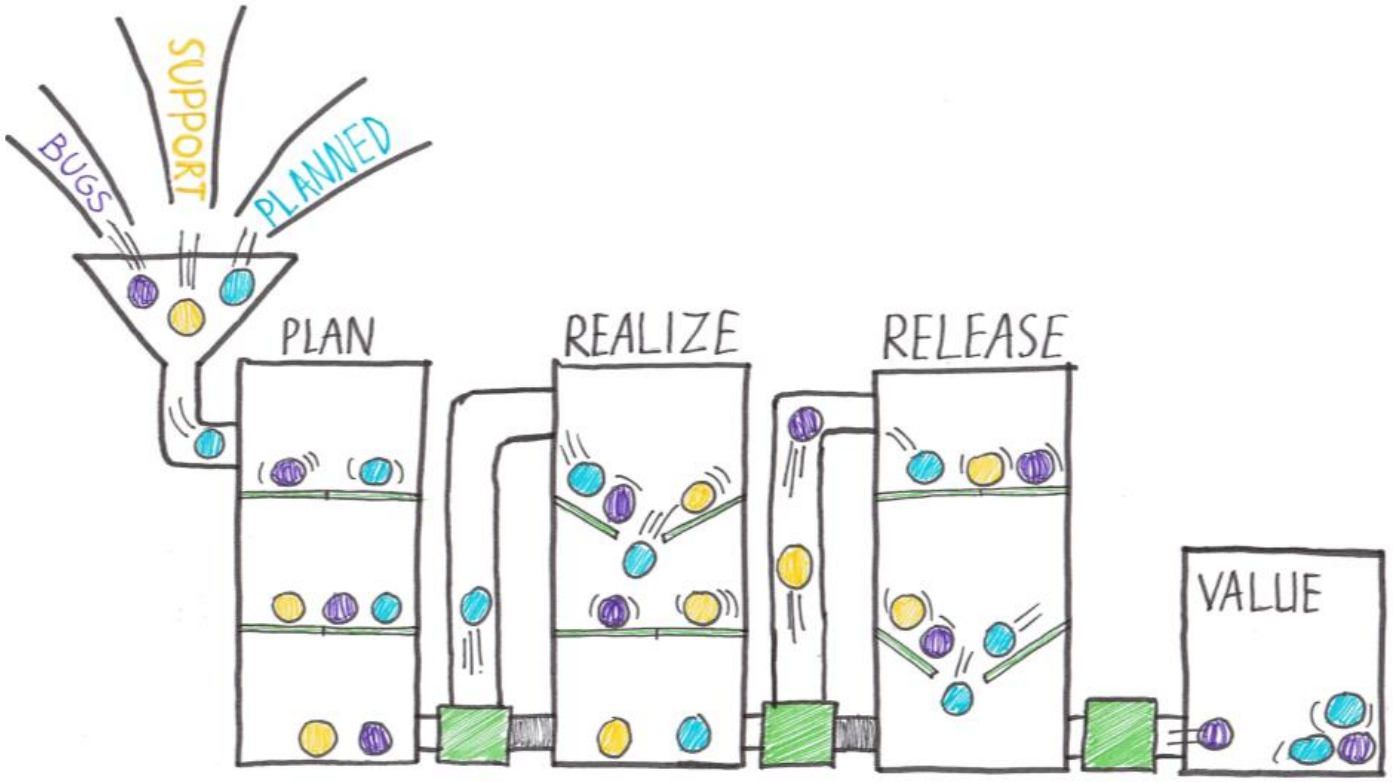
Why is this the case?

- IT should support core business and should be cheap
- Change is dangerous – should be kept under strict control
- Software is of poor quality and should be thoroughly tested
- Business works with customers and fixed functionality and time
- Computer power expensive and must be used optimally
- Customers don't want to wait for the next release so use hotfixes as a shortcut or change their requirements

The result

- Development motivated by change from business pressure
- IT motivated to change nothing to prevent introducing risk
- Result is long cycles and large batch sizes that pile on risk

Traditional IT Companies



Times have changed

Companies need to become

Better

Quality needs to be high. Applications need to be secure. Code needs to be maintainable

Faster

Customers demand features. They want it fast, otherwise they go to the competitor

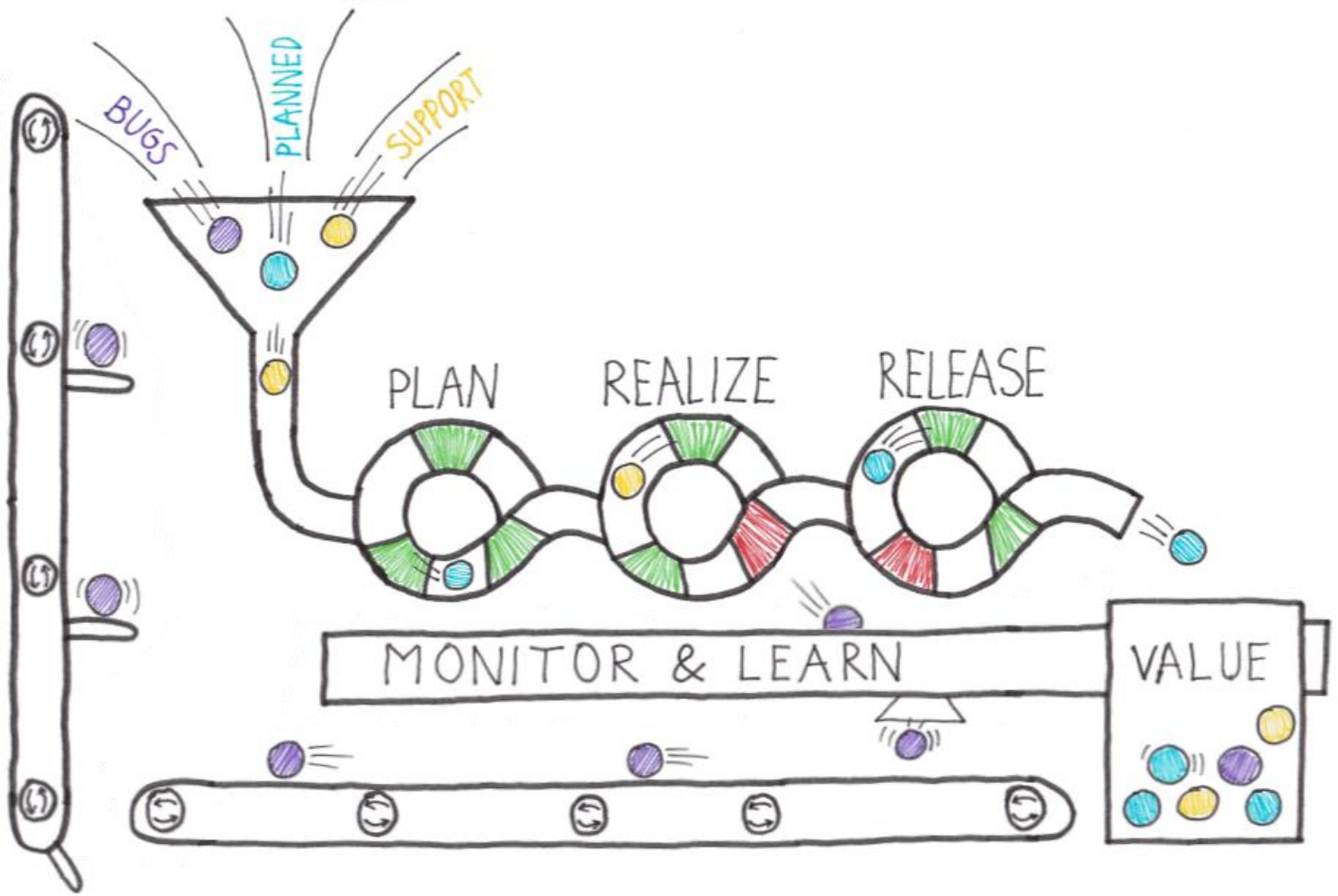
Cheaper

Competition is fierce and we are competing with the neighbor next door

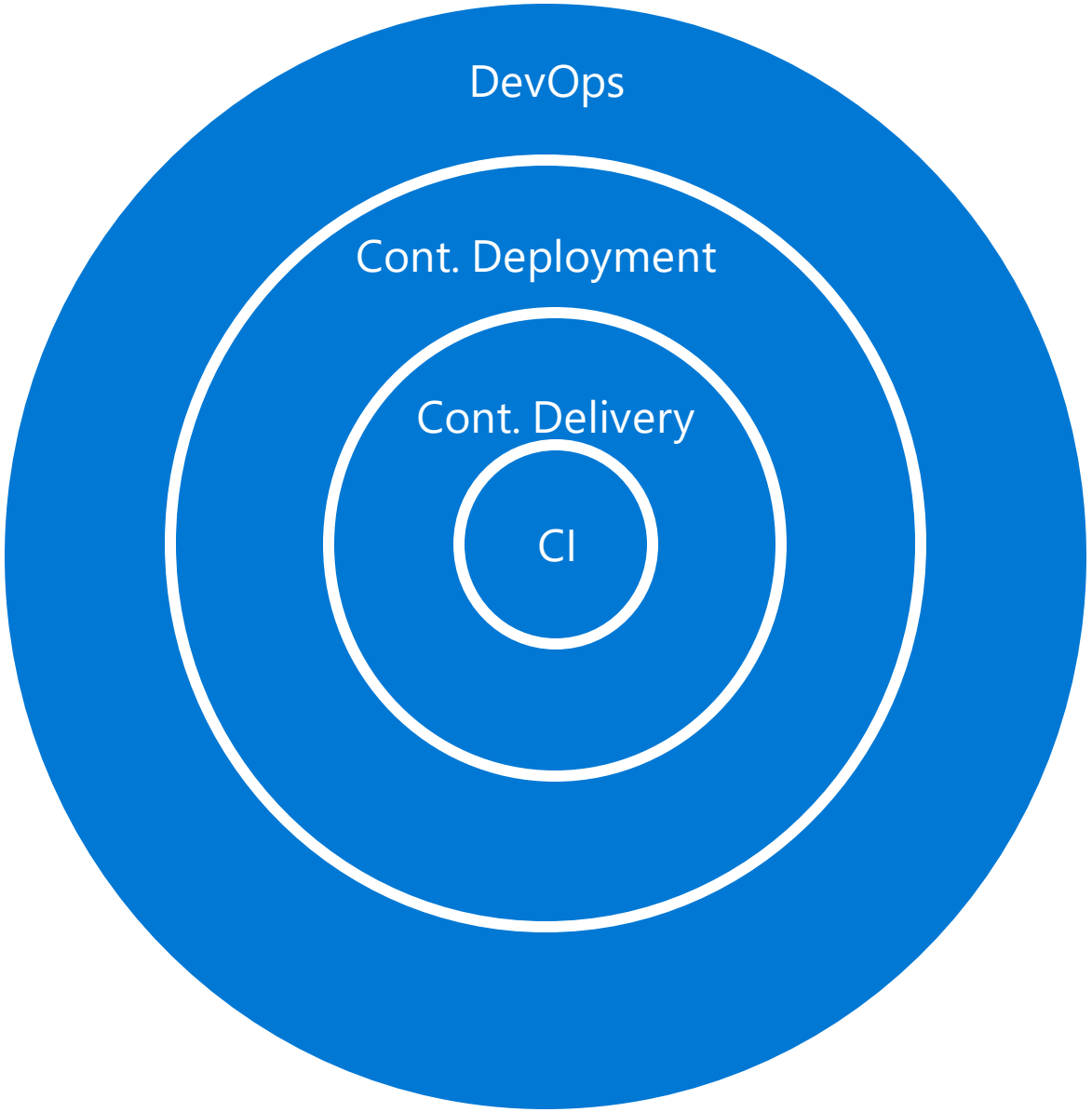
Continuous Delivery

- Continuous Delivery is delivering on the Agile Promise
- Bottlenecks shifted after implementing Agile
- Problem is within Delivery
  - Separation of Dev and Ops
  - Dev looks for change
  - Ops looks for Stability
  - This blocks delivery
- By implementing DevOps and Continuous Delivery we should be able to push out changes on demand !

Continuously delivering value



Continuous Integration, Delivery, Deployment and DevOps



Lesson 02: Release Strategy Recommendations

Release And Deployments

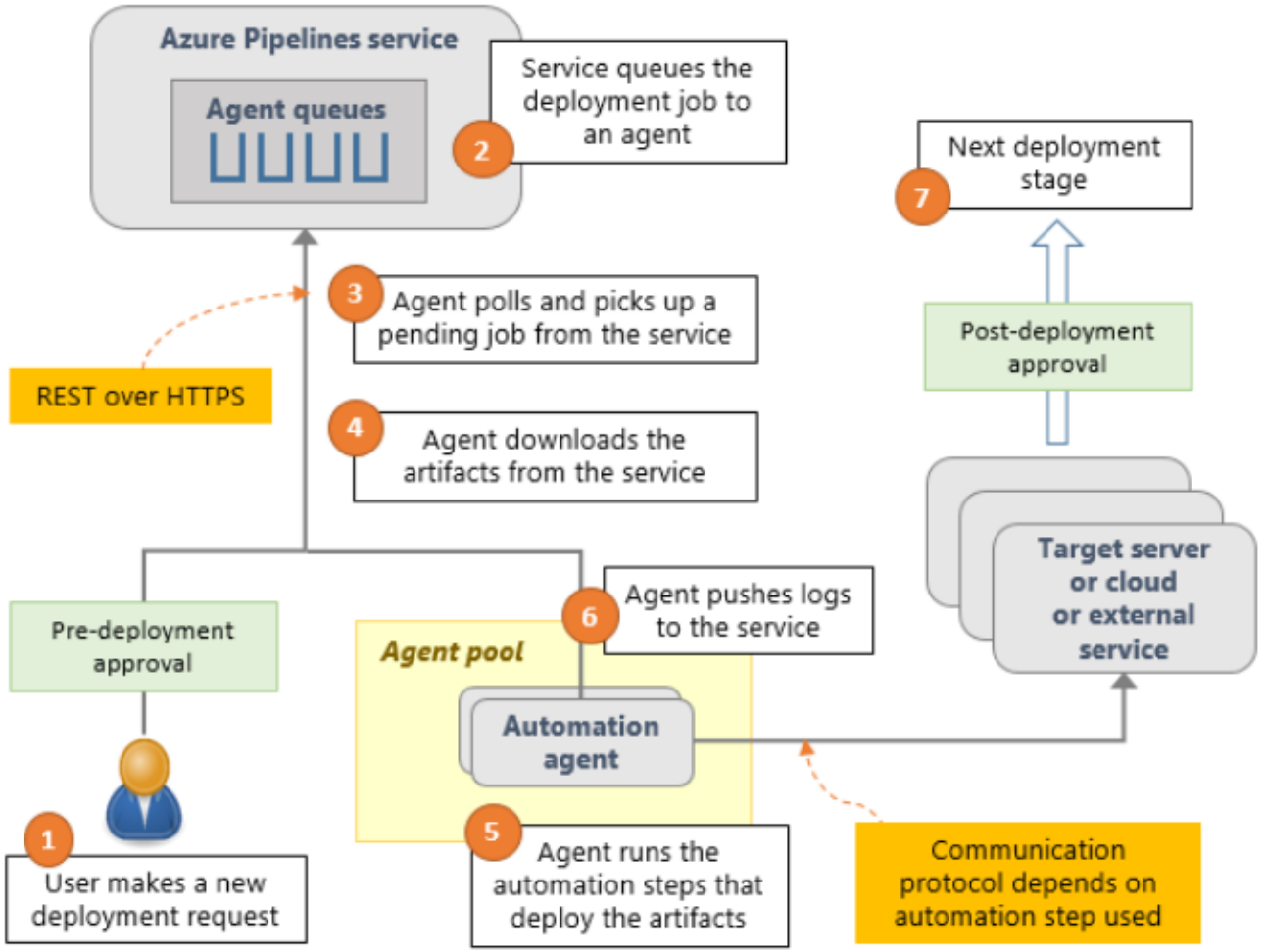
- Release and deployment are often coupled
- Release is not the same as a deployment
- Separate that in functional and technical release
  - Functional release is exposing features to customers
  - Technical release is deploying functionality

Release vs Deployment

Use similar tools but differentiate by:

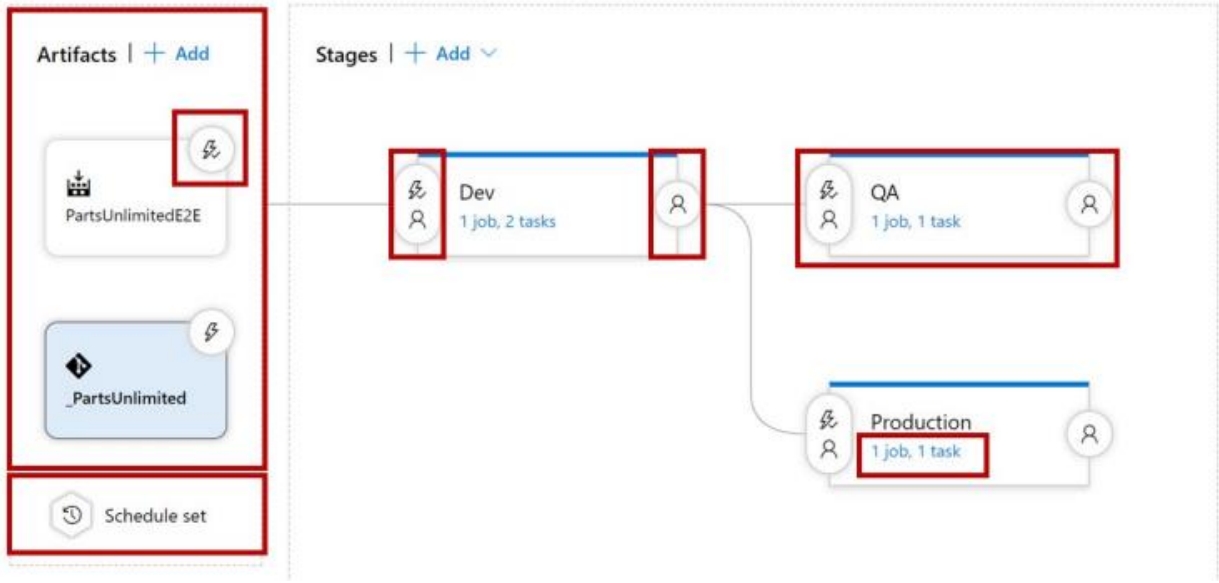
- Release:
  - Collection of versioned artifacts, pipeline, approvals, stages, variables
  - When authoring a release pipeline, you link the appropriate artifact sources to your release pipeline
- Deployment:
  - The Release provisioned to a specific infrastructure environment
  - Infrastructure can also be provisioned -> Infrastructure as code (IaC)
    - Provisioning: ARM Templates, Terraform
    - Configuration: Ansible, DSC
    - CLI, Powershell

Release in Azure DevOps



Components in a Release Pipeline

- Artifacts
- Deployment Triggers / Schedules
- Stages
- Pre- / Post Deployment Conditions
  - Approval
  - Gates

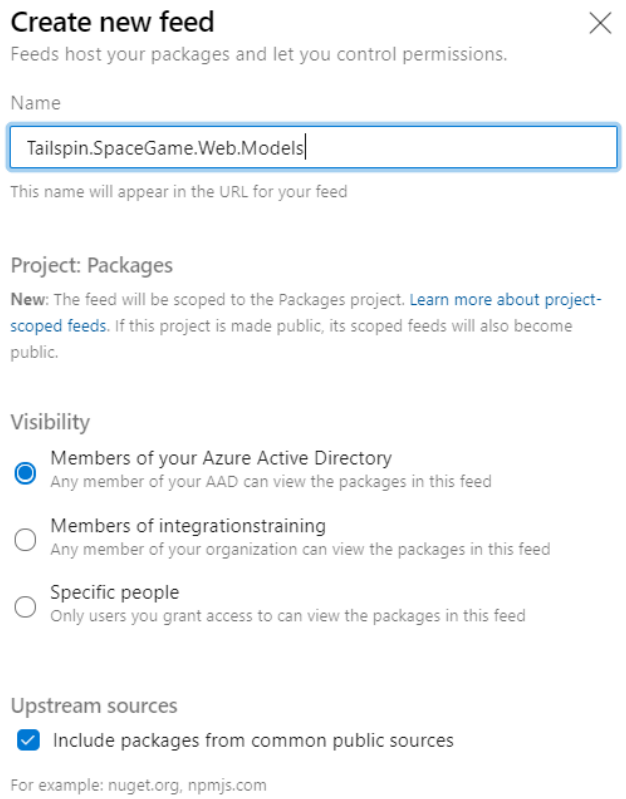


Artifacts

- Encourage the principles of build once and deploy many times
- Implemented as versioned, immutable packages
- Azure DevOps offers:
  - Releases -> Outcome of a Build
  - Artifacts -> Common Storage / Publishing Point for Packages
  - (Package) Feeds -> Logical Group of Packages

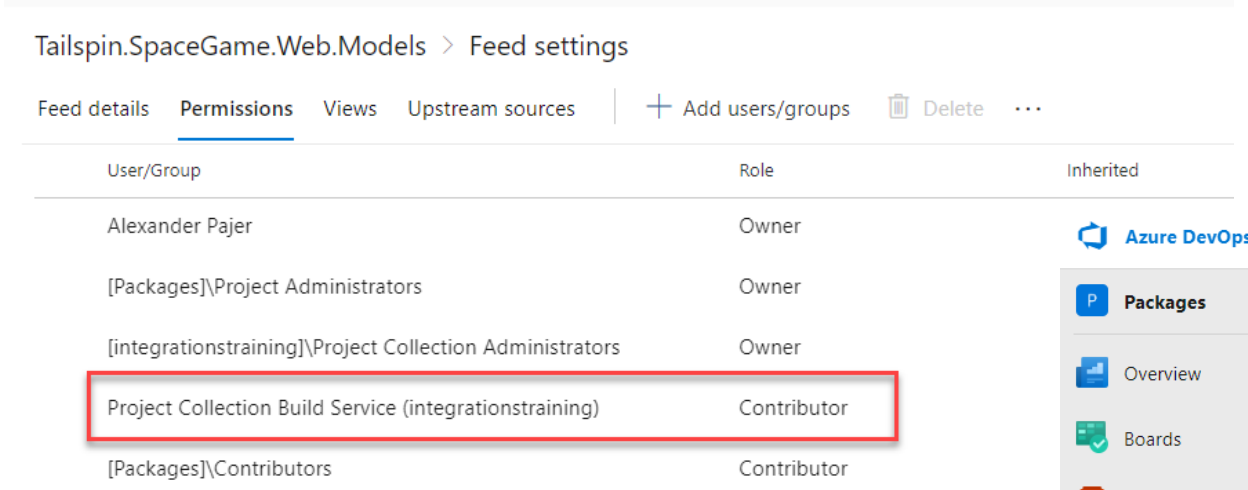
Feeds

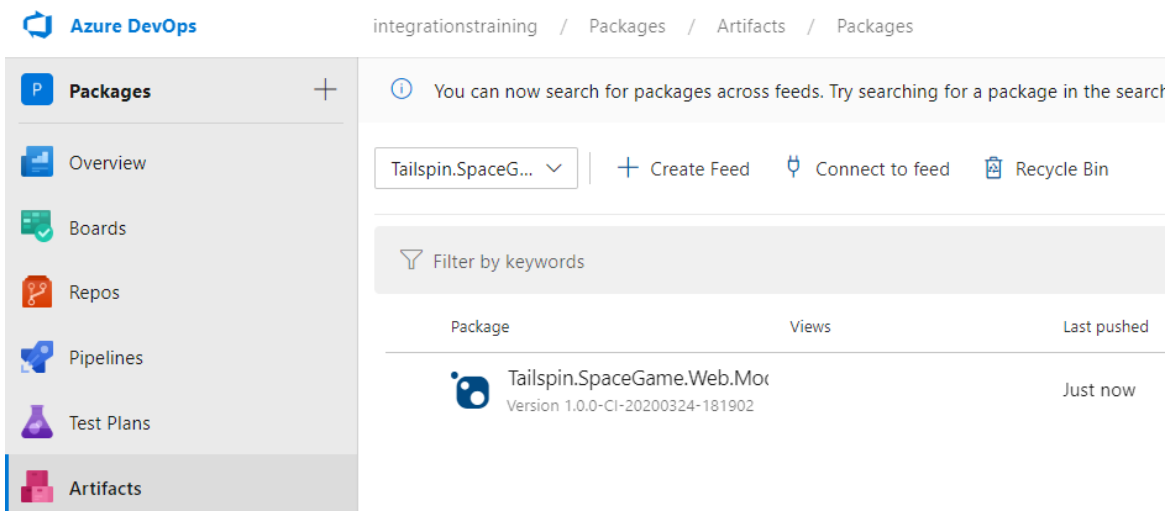
- Feeds are an organizational construct that allow you to group packages and control who has access to them with permissions.
- Ogranization:
  - Project-scoped feeds vs. Organization-scoped feeds
  - Public feeds
    - Feeds that live inside public Projects
    - Aren't intended as a replacement for existing registries of record
      - (NuGet.org, npmjs.com, etc.)



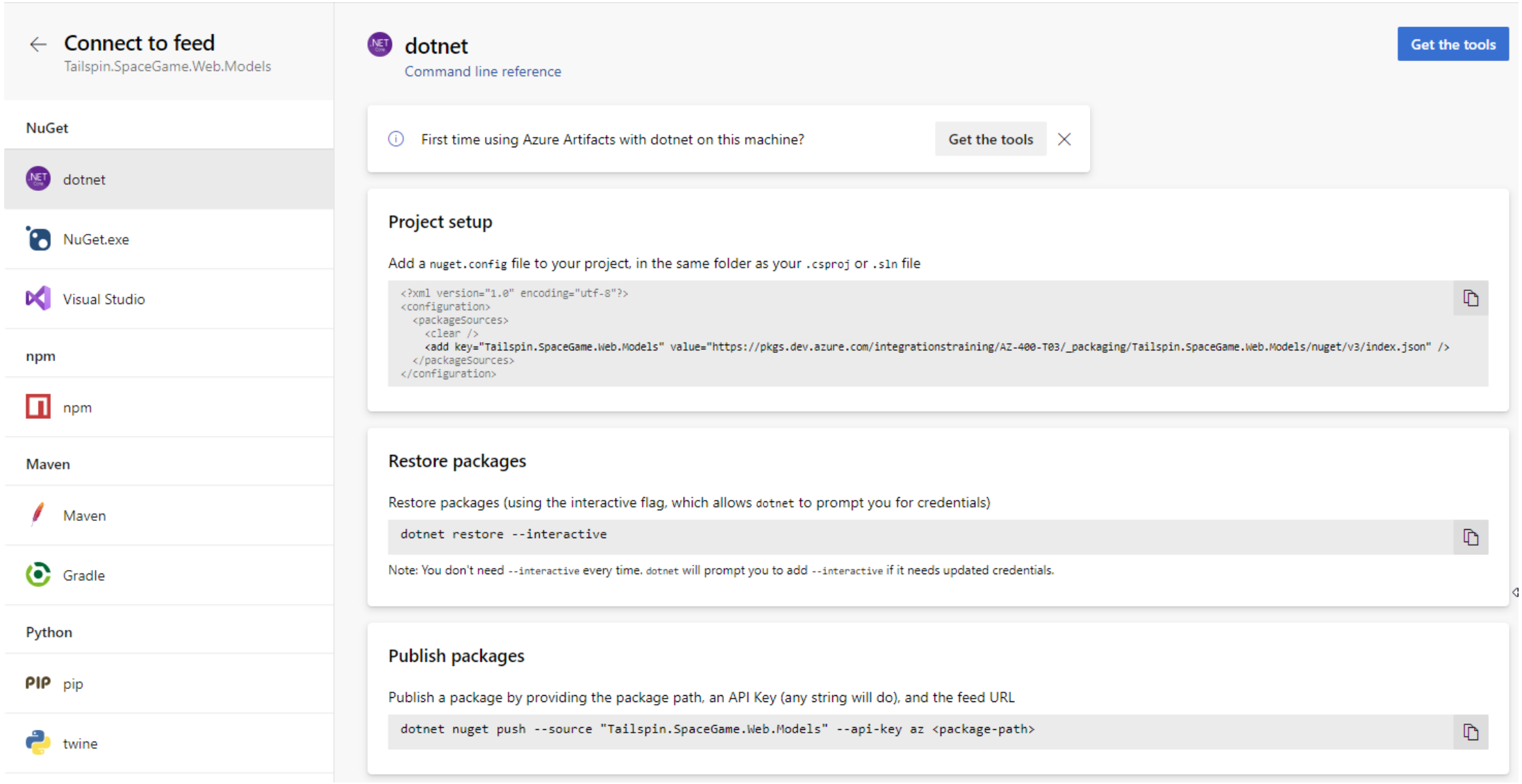
Feed Permissions

- In order to be able to publish packages to a feed one needs to have "Contribute" permissions on that feed

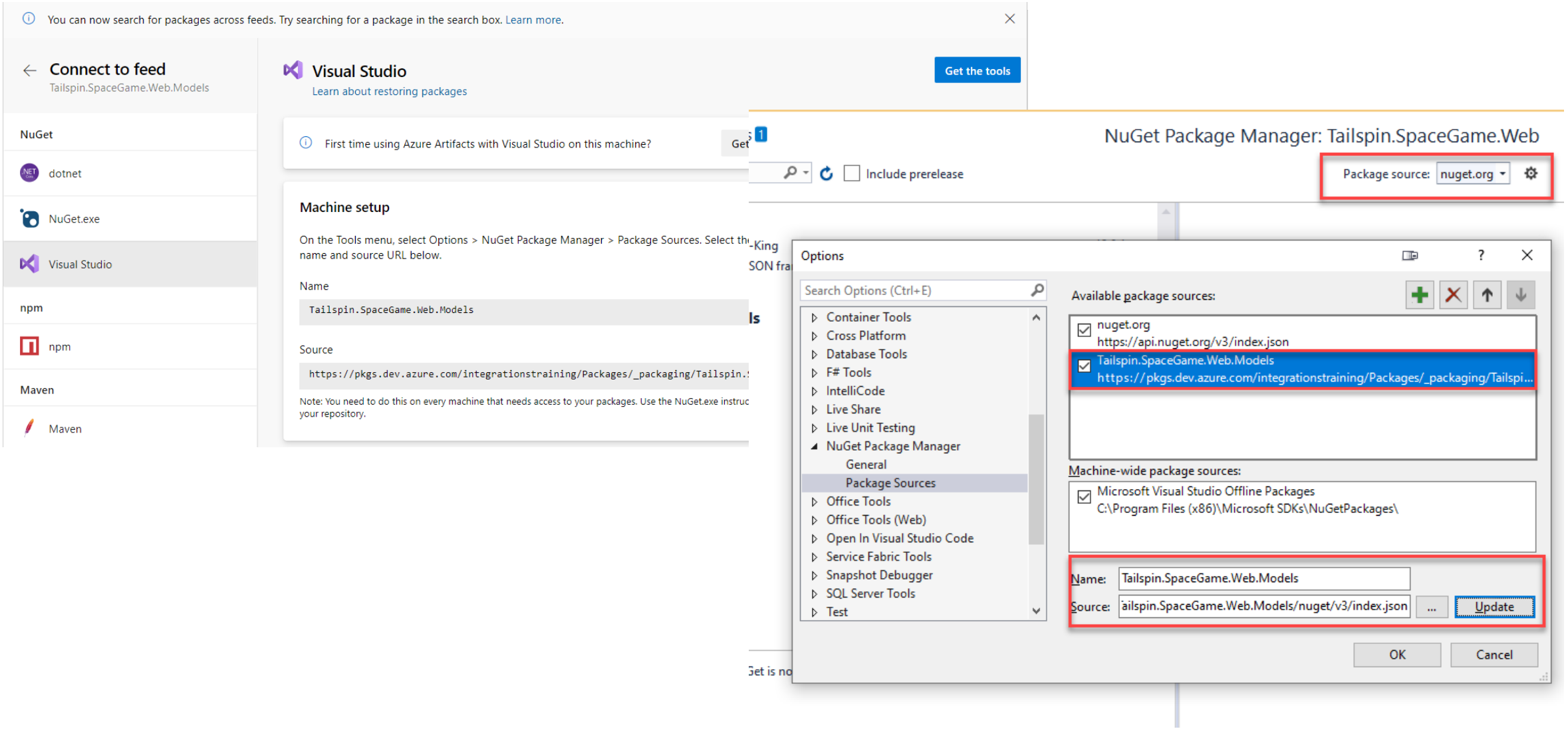




Connect to Feed VS Code

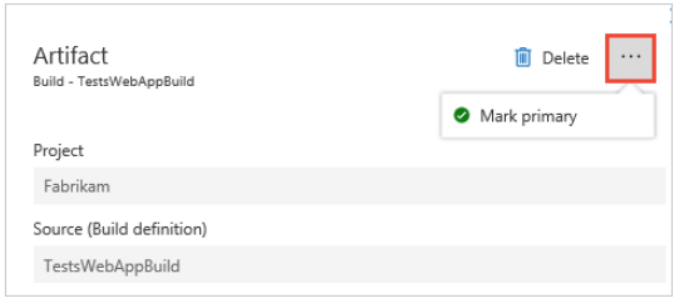


Connect to Feed VS Prof



Artifact Sources

- Define where the Artifacts come from
  - Build, Package Repos
  - Source Control / FileSystem
  - Container Registries (Dockerhub, ACR, ...)



The following sections describe how to work with the different types of artifact sources.

- Azure Pipelines
- TFVC, Git, and GitHub
- Jenkins
- Azure Container Registry, Docker, and Kubernetes
- Azure Artifacts (NuGet, Maven, npm, Python, and Universal Packages)
- External or on-premises TFS
- TeamCity
- Other sources

Packages

- Contains reusable code that other developers can use in their own projects

Package Types:

NuGet: packages .NET libraries

Maven: packages Java libraries

npm: packages JavaScript libraries

Chocolatey: packages Windows applications

RubyGems: packages Ruby libraries

Packages Hosts:

NuGet: nuget.org

Maven: apache.org

npm: npmjs.com

Chocolatey: chocolatey.org

RubyGems: rubygems.org

```
- task: NuGetCommand@2
  displayName: 'Restore project dependencies'
  inputs:
    command: 'restore'
    restoreSolution: '**/*.sln'
    feedsToUse: 'select'
    vstsFeed: 'Packages/Tailspin.SpaceGame.Web.Models'
```

Supported Source Repos

- At the beginning of each non-deployment pipeline job, the agent downloads files from your repository into a local sources directory

Repository type	Azure Pipelines (YAML)	Azure Pipelines (classic editor)	Azure DevOps Server 2019, TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Azure Repos Git	Yes	Yes	Yes	Yes
Azure Repos TFVC	No	Yes	Yes	Yes
Bitbucket Cloud	Yes	Yes	No	No
Other Git (generic)	No	Yes	Yes	Yes
GitHub	Yes	Yes	No	No
GitHub Enterprise Server	Yes	Yes	TFS 2018.2 and higher	No
Subversion	No	Yes	Yes	No

Choosing the right artifact source

- Depending on the choosen artifact source options might differ by:
  - Auto-trigger releases,
  - Trigger conditions
  - Artifact versions
  - Artifact variables
  - Work items and commits
  - Artifact download

Deployment Stages

What is a stage?

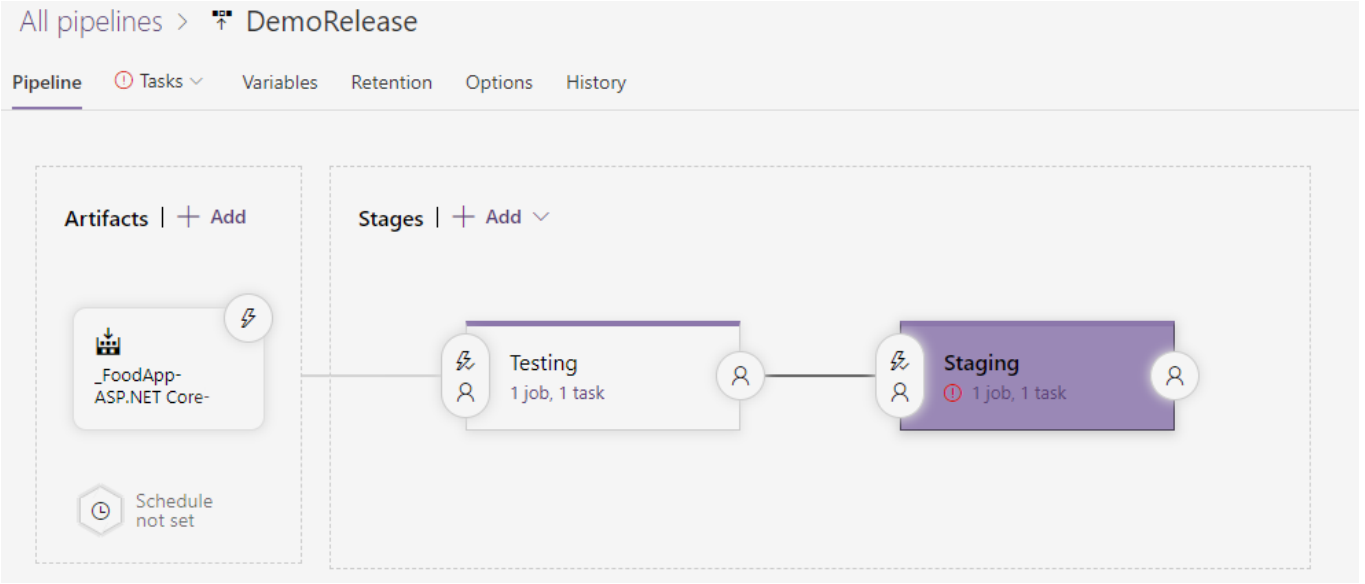
- Environment where we deploy to (Testing, Staging, Production, ...)
- Can be short or long lived

What is your target environment?

Long-lived or short-lived environments

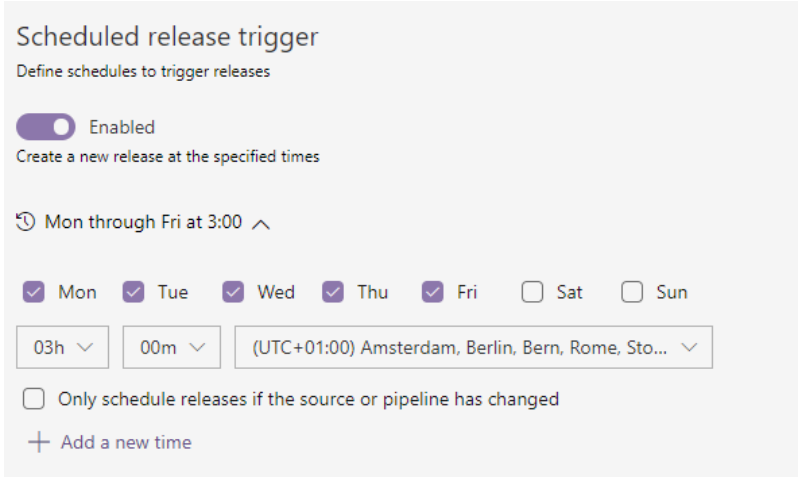
Purpose of an environment

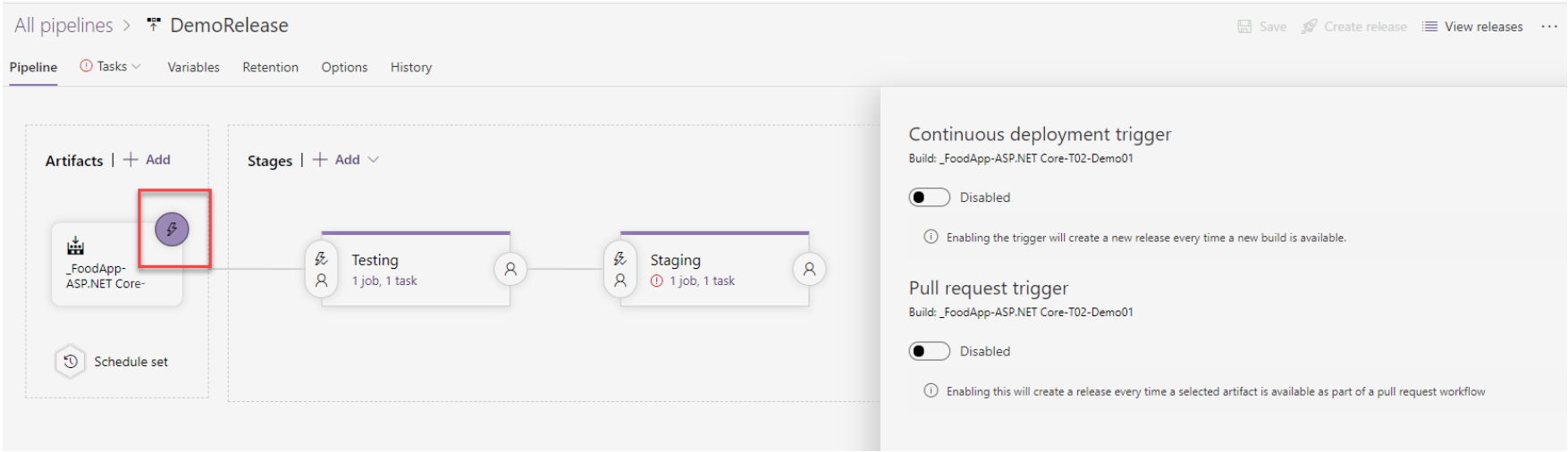
Feature release and bugfix



Release - Trigger Types

- Continuous Deployment Trigger
  - Typically triggered by a build
- Pull Request Trigger
  - … a feature branch is finished
- Scheduled Trigger
- Manual Trigger

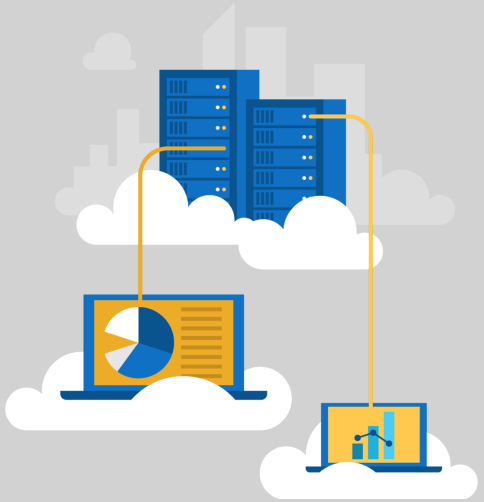




Selecting an artifact source

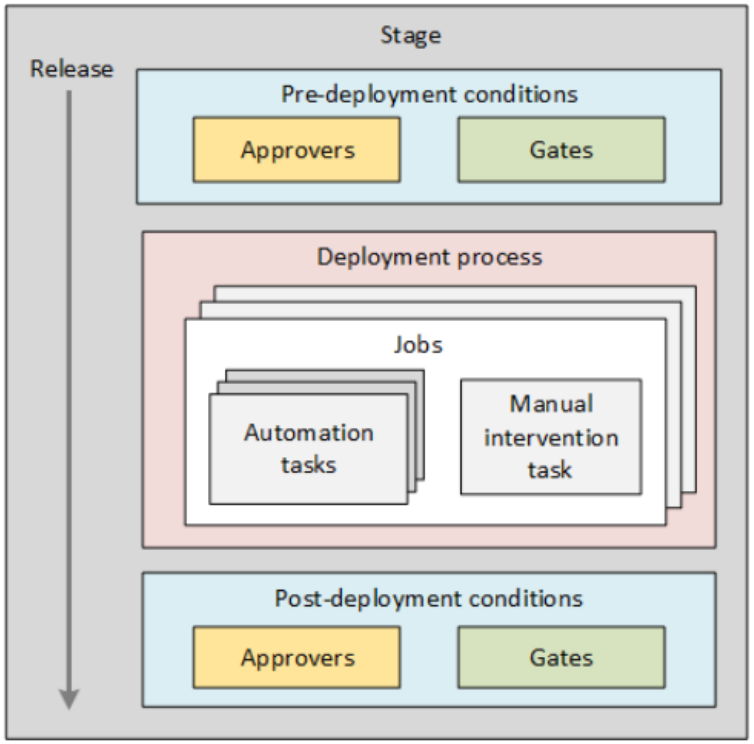
Setting up deployment stages

Lesson 03: Building a High Quality Release Pipeline



Controlling Release

- **Approvals** and **Quality Gates** give you additional control over the start and completion of the deployment pipeline



Release Approvals

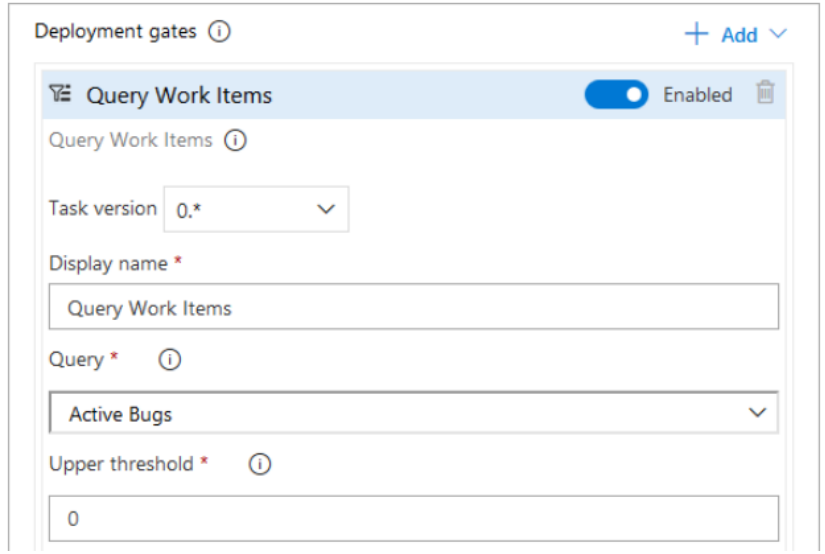
- Considerations for manual approvals
- What do we want to achieve with the approval?
- Who needs to approve?
- When do you want to approve?

Release Gates

Give you additional control over the start and completion of the deployment pipeline.

The following gates are available by default:

- Invoke Azure function
- Query Azure monitor alerts
- Invoke REST API
- Query Work items
- Security and compliance assessment



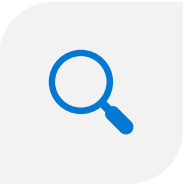
Release Gate Examples

INCIDENT AND ISSUES MANAGEMENT

NOTIFICATION OF USERS BY INTEGRATION WITH COLLABORATION SYSTEMS

QUALITY VALIDATION

SECURITY SCAN ON ARTIFACTS



USER EXPERIENCE RELATIVE TO BASELINE

CHANGE MANAGEMENT

INFRASTRUCTURE HEALTH

Setting up Manual Approvals

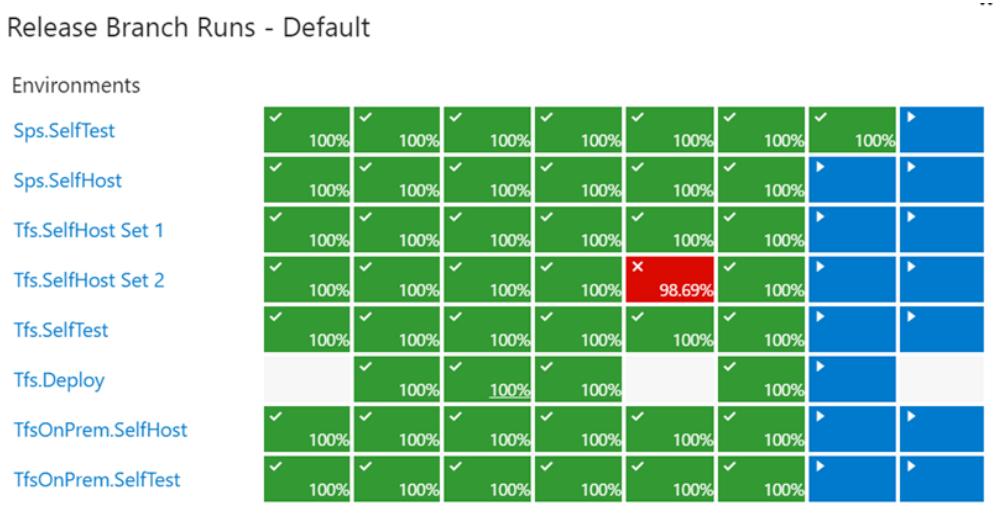
Setting up a Release Gate

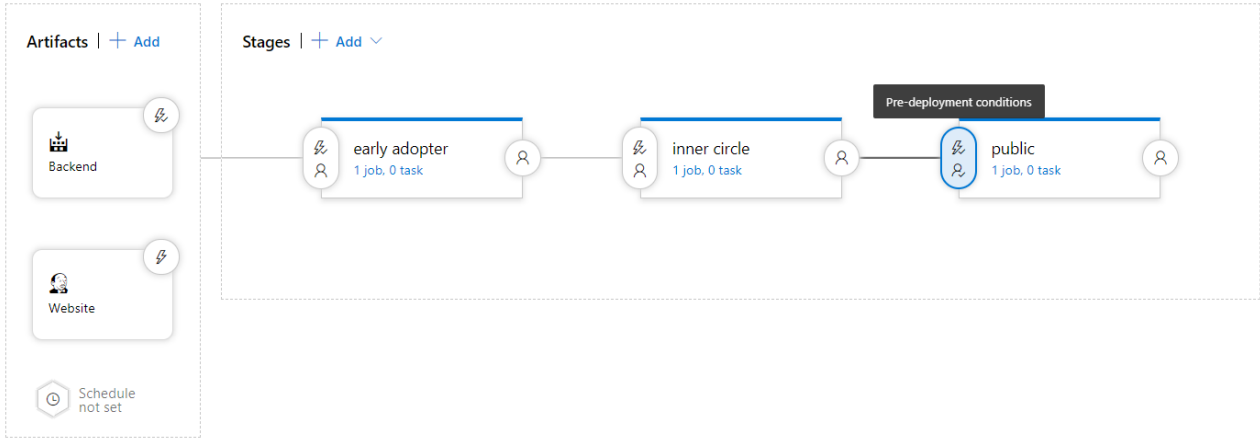
How to measure quality of your release process

Visualize your release process

Symptoms of broken process (every second day, only after rerun, never ending up in last stage)

Dashboard widgets





Quality Gates

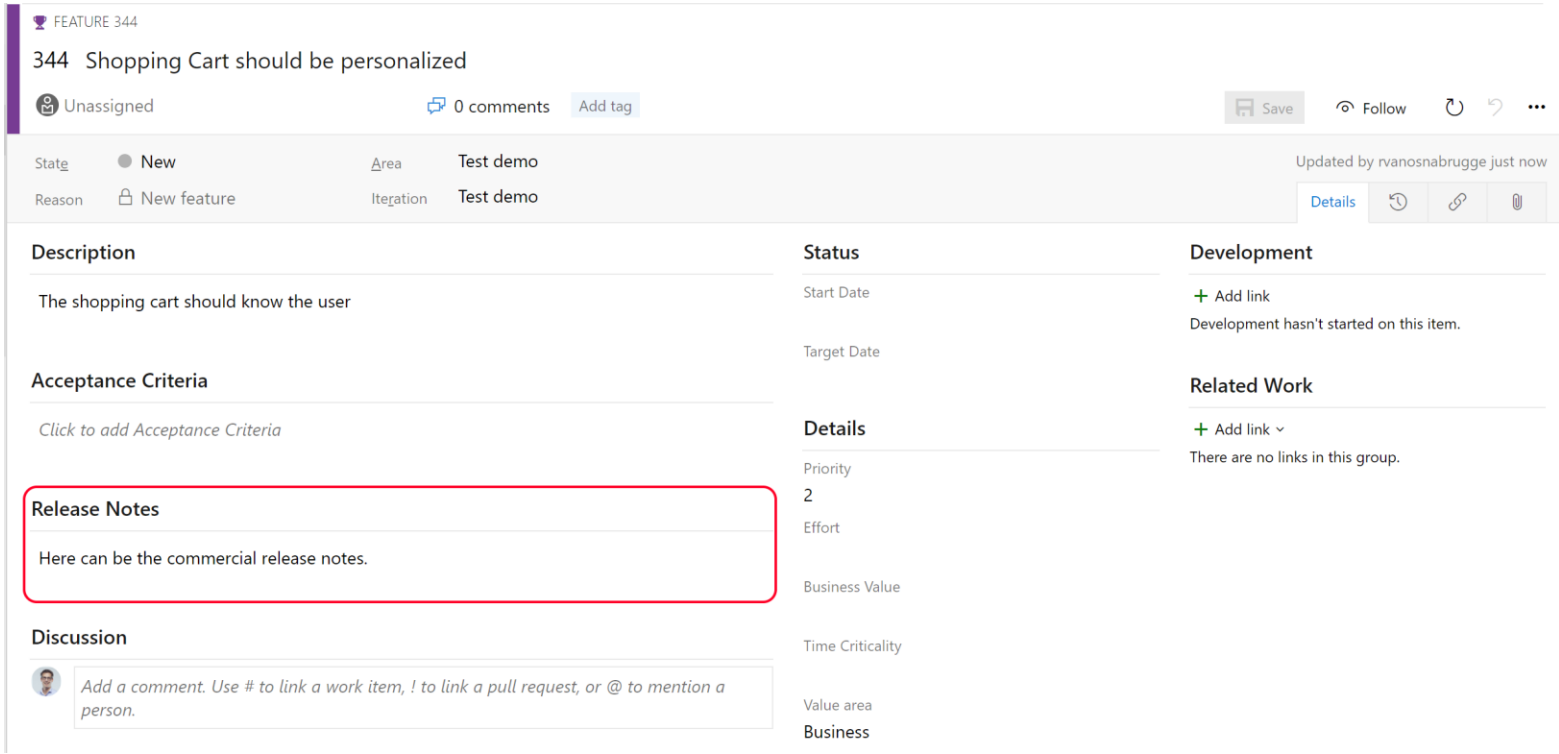
A quality gate is the best way to enforce a quality policy in your organization. It's there to answer one question: can I deliver my application to production or not?

Using Release Gates to protect quality

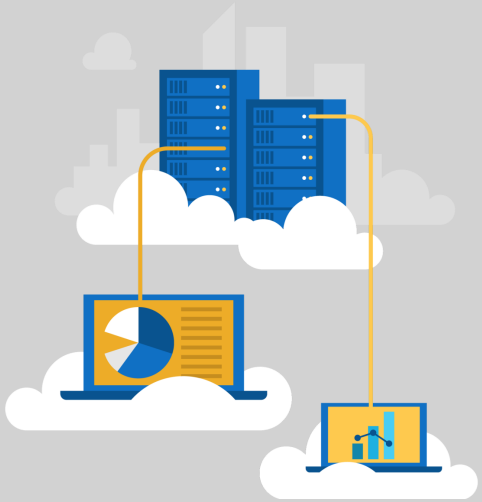
- No new blocker issues
- Code coverage on new code greater than 80%
- No license violations
- No vulnerabilities in dependencies
- No new technical debt introduced
- Compliance checks
- Are there work items linked to the release?
- Is the release started by someone else as the code committer?
- Is the performance not affected after a new release?

Release notes and Documentation

- Technical or Functional Documentation?
- Where to store Documentation
  - Document Store
  - Wiki
  - In the code base
  - In a Work Item



Lesson 04: Choosing a Deployment Pattern

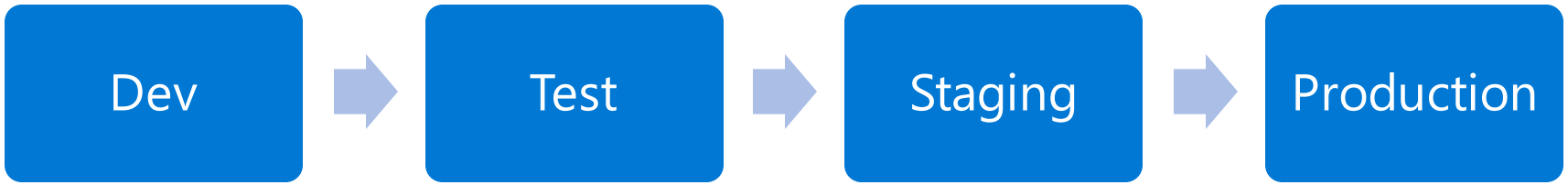


Deployment patterns

A deployment pattern is a way of how you choose to move your application to production.

Traditional Deployment Patterns (DTAP)

Modern Deployment Patterns



Modern Deployment Patterns

- Blue-Green Deployment
  - Two production environments
- Canary Release
  - Release is only pushed to a small number of users
- Dark Launching
  - Releasing production-ready features to a subset of your users before a full release
- Progressive Exposure Deployment
  - Expose the new software to a subset of users, that you extend gradually over time

Release Management Tools

- Tools that can do Build and Continuous Integration and Deployment
- Tools that can do Release Management

Artifacts and Artifact source

Triggers and Schedules

Approvals and gates

Stages

Build and Release Tasks

Traceability, Auditability and Security

Release Management Tools

- Jenkins
- Circle CI
- Azure DevOps Pipelines
- GitLab Pipelines
- Atlassian Bamboo
- XL Deploy/XL Release

GitLab

Azure Pipelines

Jenkins

Wrap up

- Differentiate between a release and a deployment
- Define the components of a release pipeline
- Explain things to consider when designing your release strategy
- Classify a release versus a release process, and outline how to control the quality of both
- Describe the principle of release gates and how to deal with release notes and documentation
- Explain deployment patterns, both in the traditional sense and in the modern sense
- Choose a release management tool

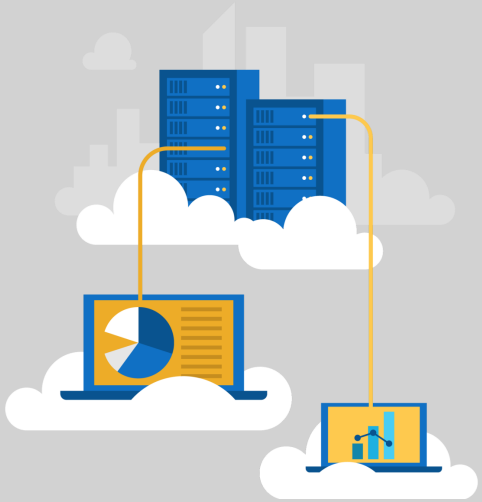
Microsoft

AZ-400.3
Module 2: Set Up a Release Management Workflow

Learning Objectives

- You are familiar with the terminology used in Azure DevOps and other Release Management Tooling
- You know what a Build and Release task is, what it can do, and are familiar with some available deployment tasks
- You know what an Agent, Agent Queue and Agent Pool is and what an agent is all about
- You know what a release job is, and why you sometimes need multiple release jobs in one release pipeline
- You know the difference between multi-agent and multi-configuration release job
- You know what release variables are, what stage variables are and how to best use them in your release pipeline
- You know what a Service Connection is and how you can use them to deploy to an environment securely
- You know what the possibilities for testing are in the pipeline and how to embed testing in the pipeline
- You are familiar with the different ways to inspect the health of your pipeline and release by using, alerts, service hooks and reports
- You know what a release gate is and how to create one

Lesson 01: Create a Release Pipeline

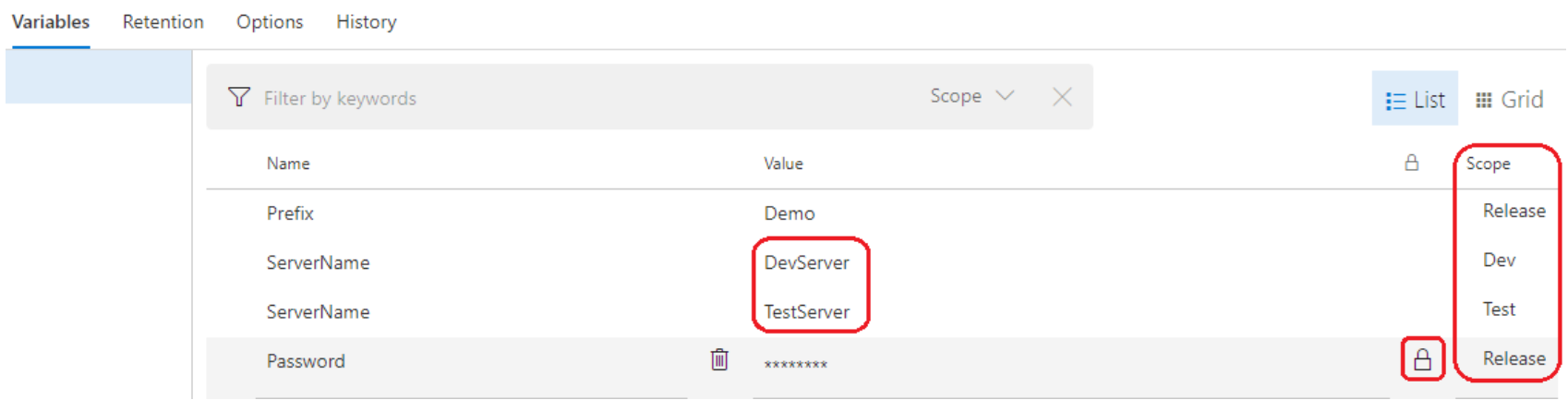


Definitions and Glossary

Term	Description	Synonym
Stage	An isolated and independent target for deployment	Environment
Job	A phase in the release pipeline that can run simultaneously with other phases on different Operating Systems	Phases
Agent	The program thaat runs the build or release	
Build & Release Task	Tasks are units of executble code used to perform designated actions in a specified order.	Action, Plugin, App
Release pipeline	The process that runs when deploying an artifact. Including triggers, approvals and gates	Release process, pipeline, release definition
CI/CD	Continuous Integration / Continuous Deployment	
Release Gate	An automated check that approves the continuation	Quality Gate, Automatic Approval
Service Connection	A secure connection to an environment or service	Service Endpoint

Release Variables

- Predefined Variables
- Release pipeline variables
- Stage Variables
- Variable groups
- Normal and Secret variables



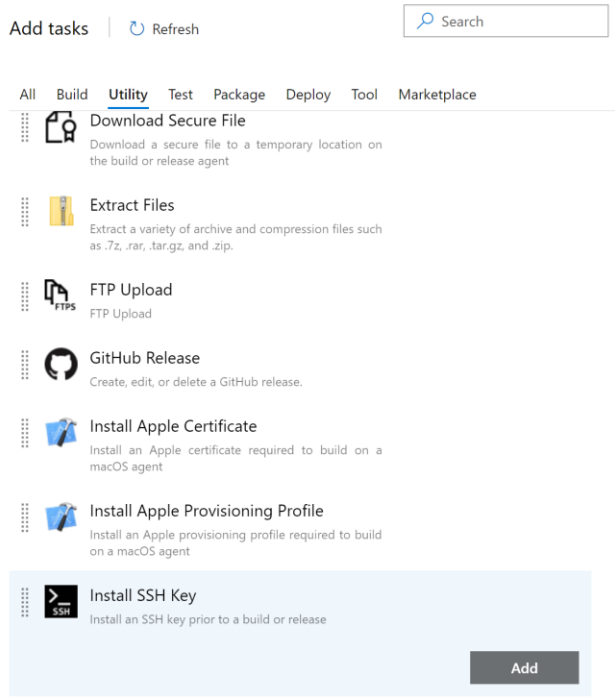
Build and Release Tasks

Units of executable code used to perform designated actions in a specified order

Tasks building, testing, running utilities, packaging, and deploying

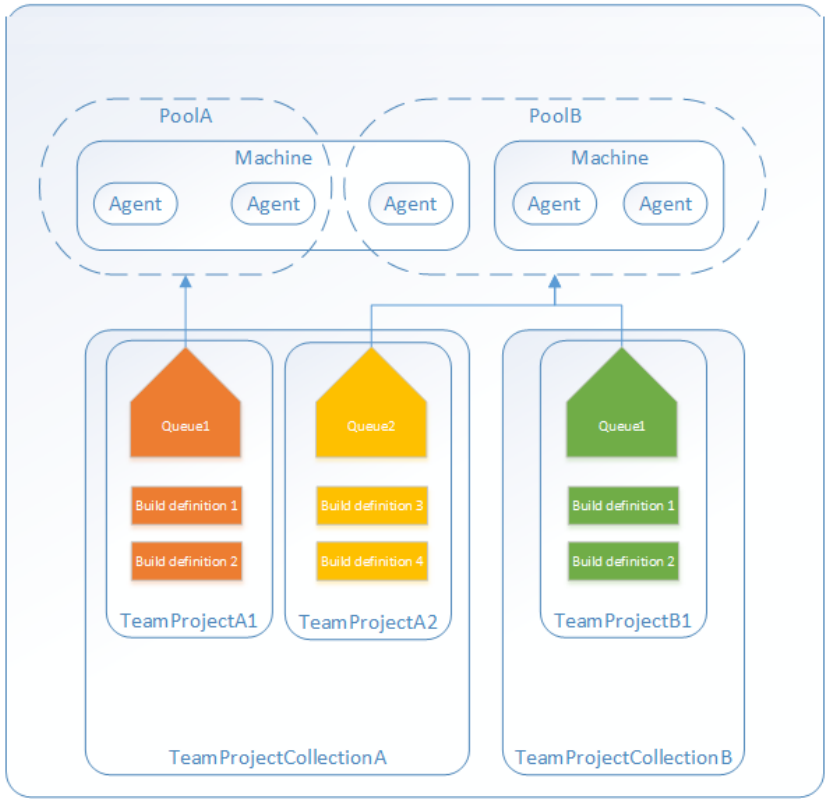
Extensible model

Community tasks available in marketplace



Agents, Agent Pools and Queues

- Agents are tasked with performing the builds and releases
- Agent pools are used to organize and define permission boundaries around your agents
- Agent queue provides access to a pool of agents.



Hosted Agents and Private Agents

Hosted agents. These agents exist within their own hosted pool and are maintained and upgraded by the vendor.

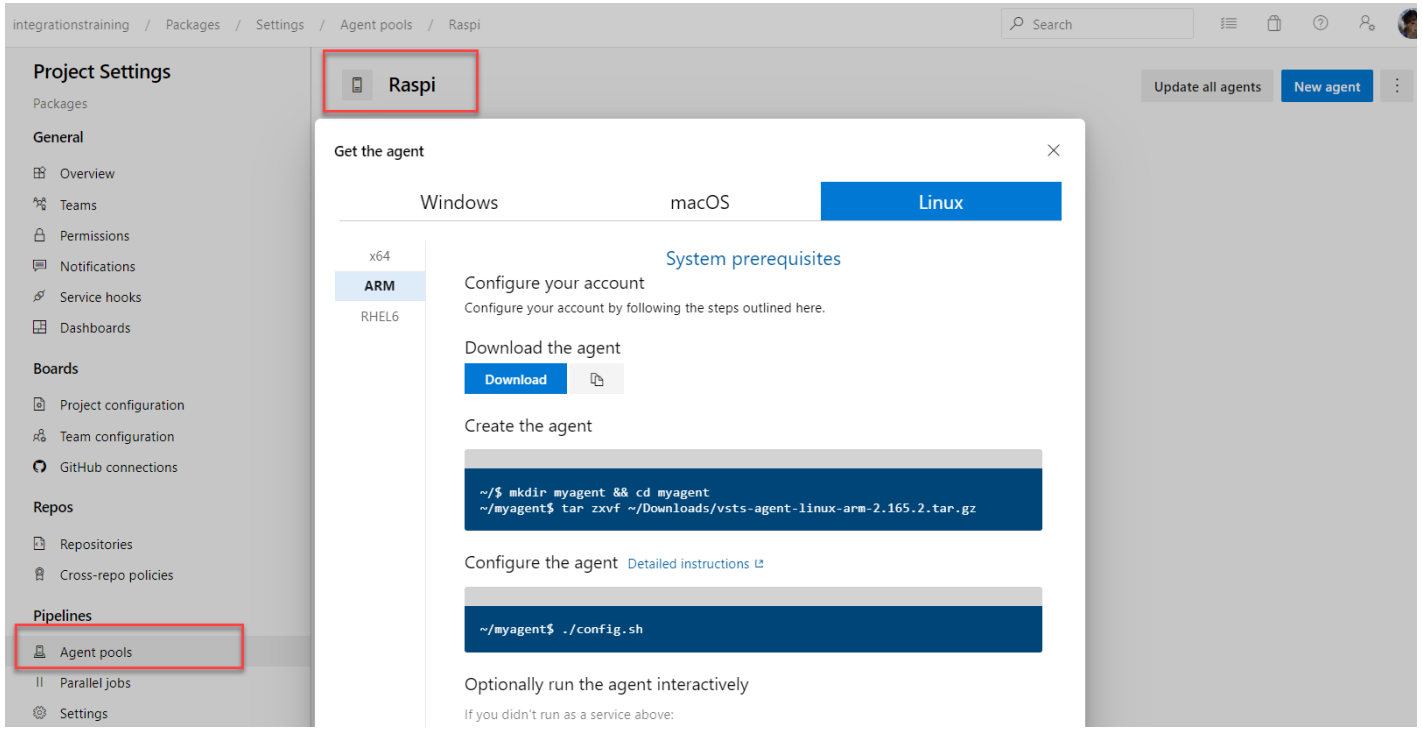
Hosted agents have specific limitations and advantages:

- Hosted agents have no cost and are immediately available, and have most common software and libraries installed.
- Do not have an interactive mode.
- Do not allow administrative privilege or allow logon.

Private (or Custom) agents. Private agents are provisioned on private virtual machines (VMs) and are custom built to accommodate the project's needs.

Agent Pools

Custom Agent Pools groups of agents that run on a separate server and install and execute actions on a deployment target



Deployment Groups

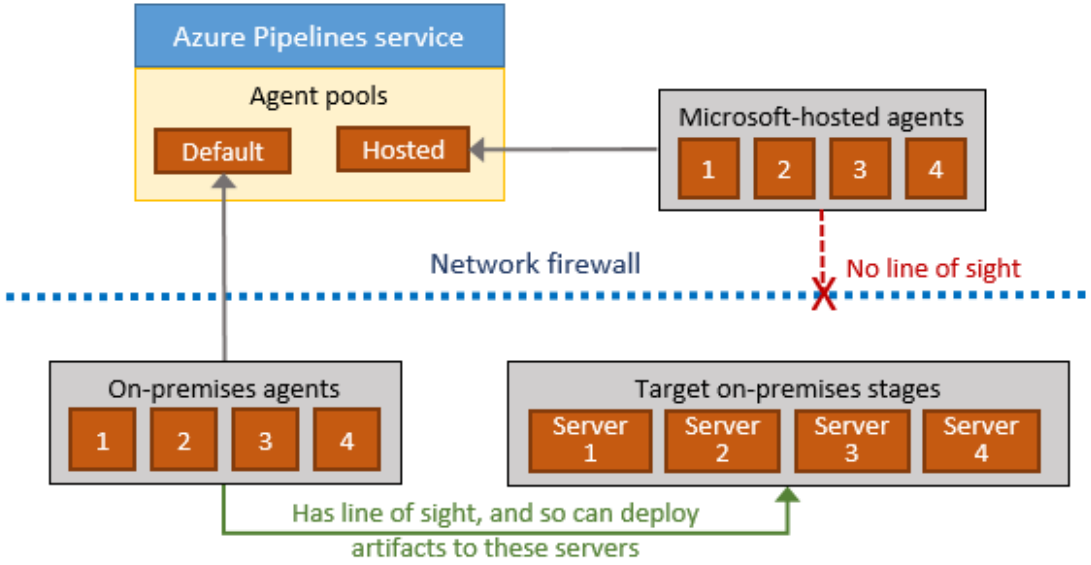
A deployment group is a logical set of deployment target machines that have agents installed on each one.

Can solve the firewall issues because you pull sources from a central server

Specify the security context and runtime targets for the agents.

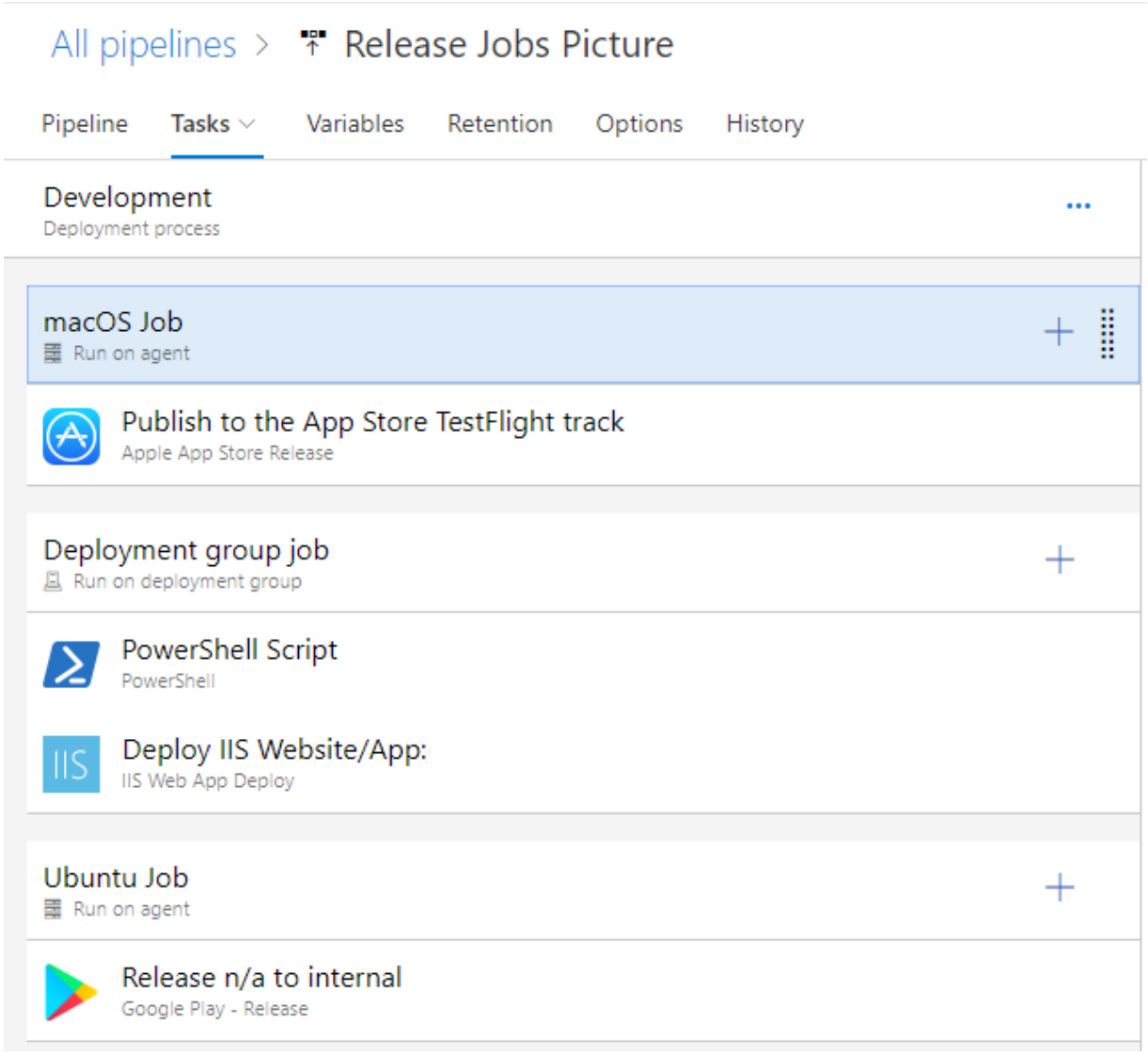
Let you view live logs for each server as a deployment takes place.

Enable you to use machine tags to limit deployment to specific sets of target servers.



Release Agent Jobs

- A job is a series of tasks that run sequentially on the same target
- Can be combined in one pipeline to enable multi-platform deployment
  - E.g. Deploy .Net backend via Windows, IoS app via MacOS and Angular frontend via Linux
- Jobs run on the host machine where the agent is installed



Different Agent Job Types

- Jobs or Agent Jobs
  - A series of tasks that run sequentially on the same target
  - When the target is an agent, the tasks are run on the computer that hosts the agent
- Server or Agentless Jobs
  - A server job does not require an agent or any target computers
- Container Jobs
  - When you specify a container in your pipeline, the agent will first fetch and start the container
- Deployment Group Jobs

Multi-Agent and Multi-configuration

Multi-configuration: Run the same set of tasks on multiple configurations

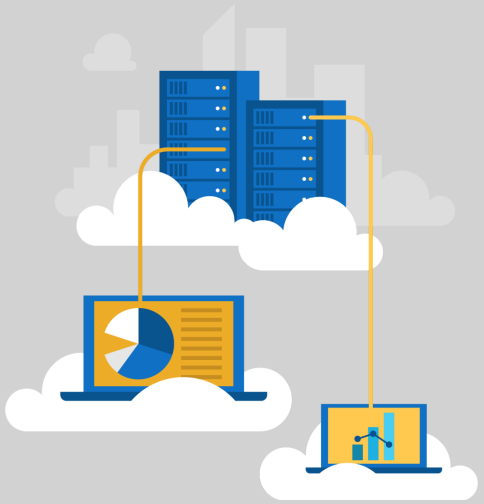
- Run the release once with configuration Setting A on WebApp A and setting B for WebApp B
- Deploy to different geographic regions.
- Multi-configuration testing: run a set of tests in parallel - once for each test configuration.

Multi-agent: Run the same set of tasks on multiple agents using the specified number of agents

- Deploy same bits to a farm of servers

Deploying to Azure VM using Deployment Groups

Lesson 03: Manage Secrets with the Release Pipeline



Manage And Modularize Tasks and Templates

Within the Azure DevOps suite, three important concepts enable reusability.

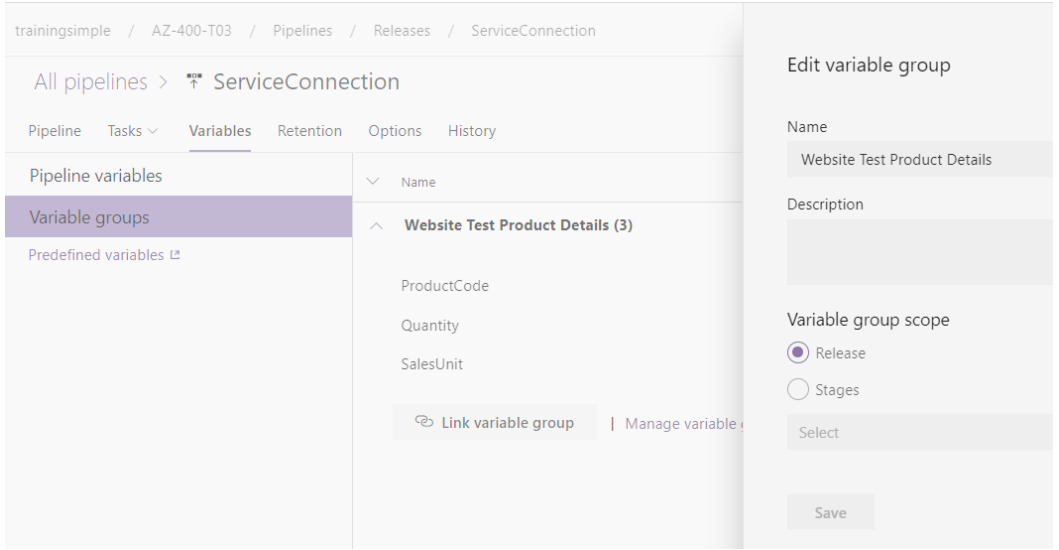
- Task Groups
- Variable Groups
- Custom Build and Release Tasks

Variable Groups

A variable group is used to store values that you want to make available across multiple builds and release pipelines.

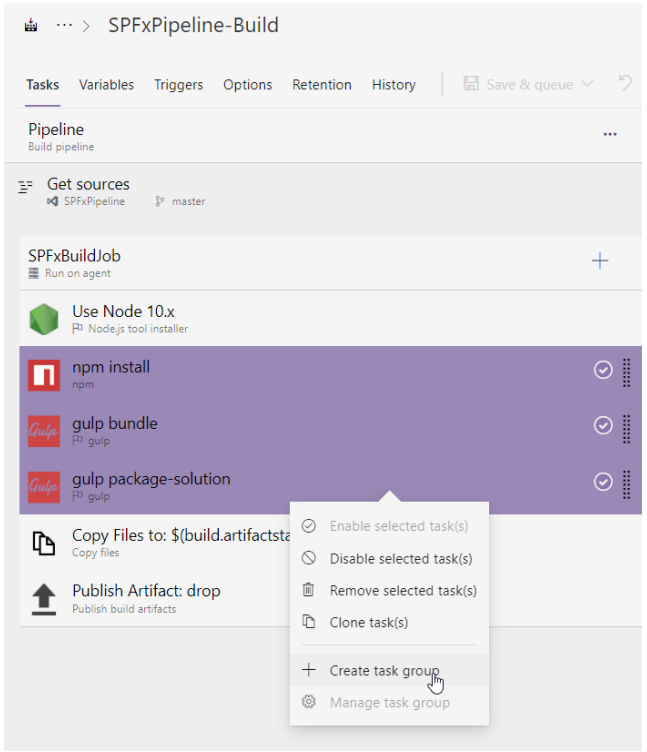
Can be used to:

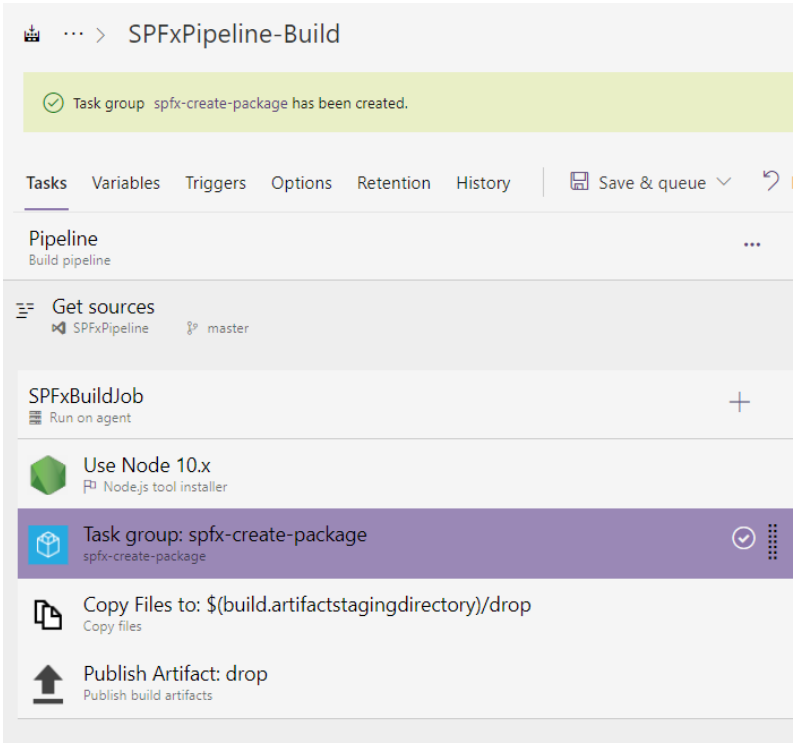
- Store the username and password for a shared server
- Store a share connection string
- Store the geolocation of an application
- Store all settings for a specific application



Task Groups

- Encapsulate a sequence of tasks in one reusable task (Templates when using yaml)
- Parameterization of task group to make reuse easier
- Standardize and centrally manage deployment steps for all your applications





YAML Templates

Provide Task Group Functionality for YAML

Support four Kinds of Templates:

- Stage
- Job
- Step
- Variable

Azure-pielines.yml references another physical file

```yaml
# File: azure-pipelines.yml

jobs:
- template: jobs/build.yml  # Template reference
  parameters:
    name: macOS
    pool:
      vmImage: 'macOS-10.14'
```

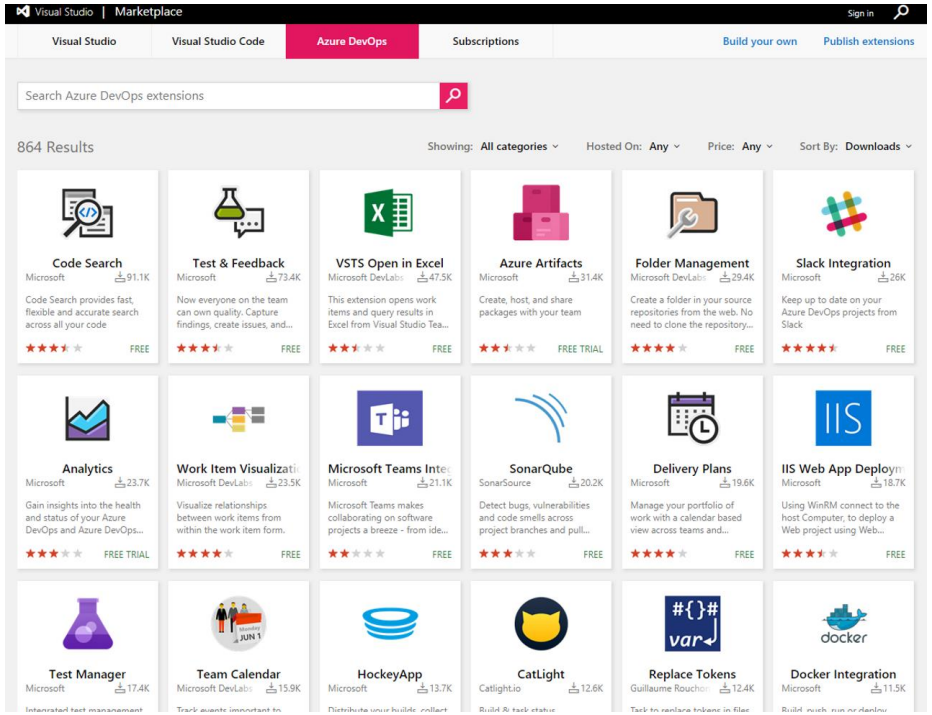
```yaml
# File: jobs/build.yml

parameters:
  name: ''
  pool: ''
  sign: false

jobs:
- job: ${{ parameters.name }}
  pool: ${{ parameters.pool }}
  steps:
  - script: npm install
  - script: npm test
  - ${{ if eq(parameters.sign, 'true') }}:
    - script: sign
```

Custom build/release tasks

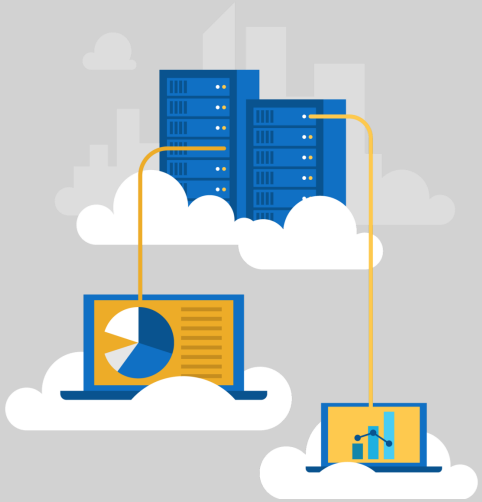
- Private or public accessible
- Access to variables that are otherwise not accessible
- Use and reuse secure endpoint to a target server
- Safely and efficiently distribute across your whole organization
- Users do not see implementation details.



Demo Creating and managing variable groups

Creating and managing task groups

Lesson 04: Integrate Secrets with the Release Pipeline



Integrate Secrets with the Release Pipeline

When you deploy your applications to a target environment, there are almost always secrets involved

- Secrets to access the target environment
  - Servers, Storage accounts
- Secrets to access resources
  - Connections strings, Tokens, username/passwords)
- Secrets that your application uses
  - Config files

Secrets in your release pipeline

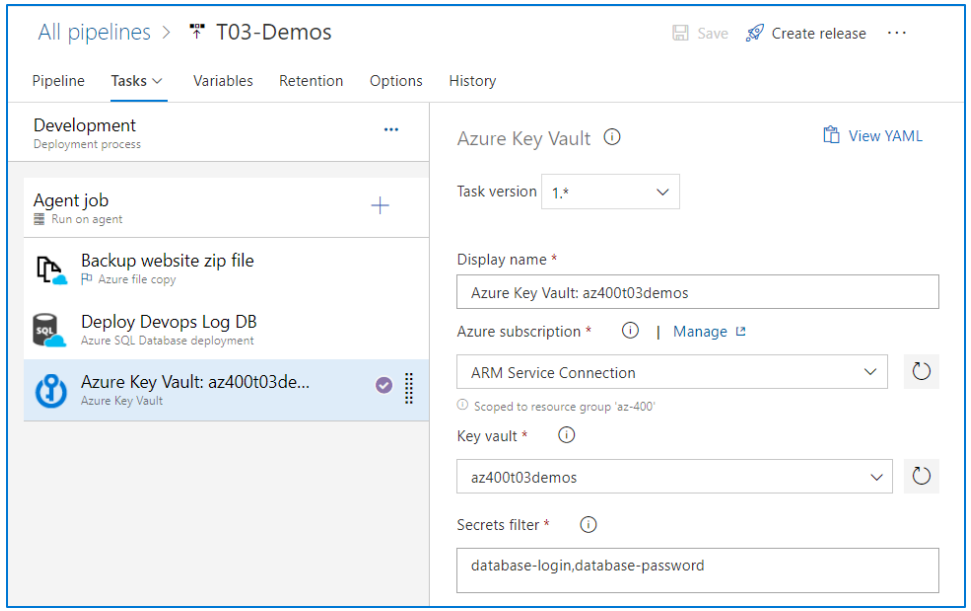
There are different ways to deal with secrets in your pipelines:

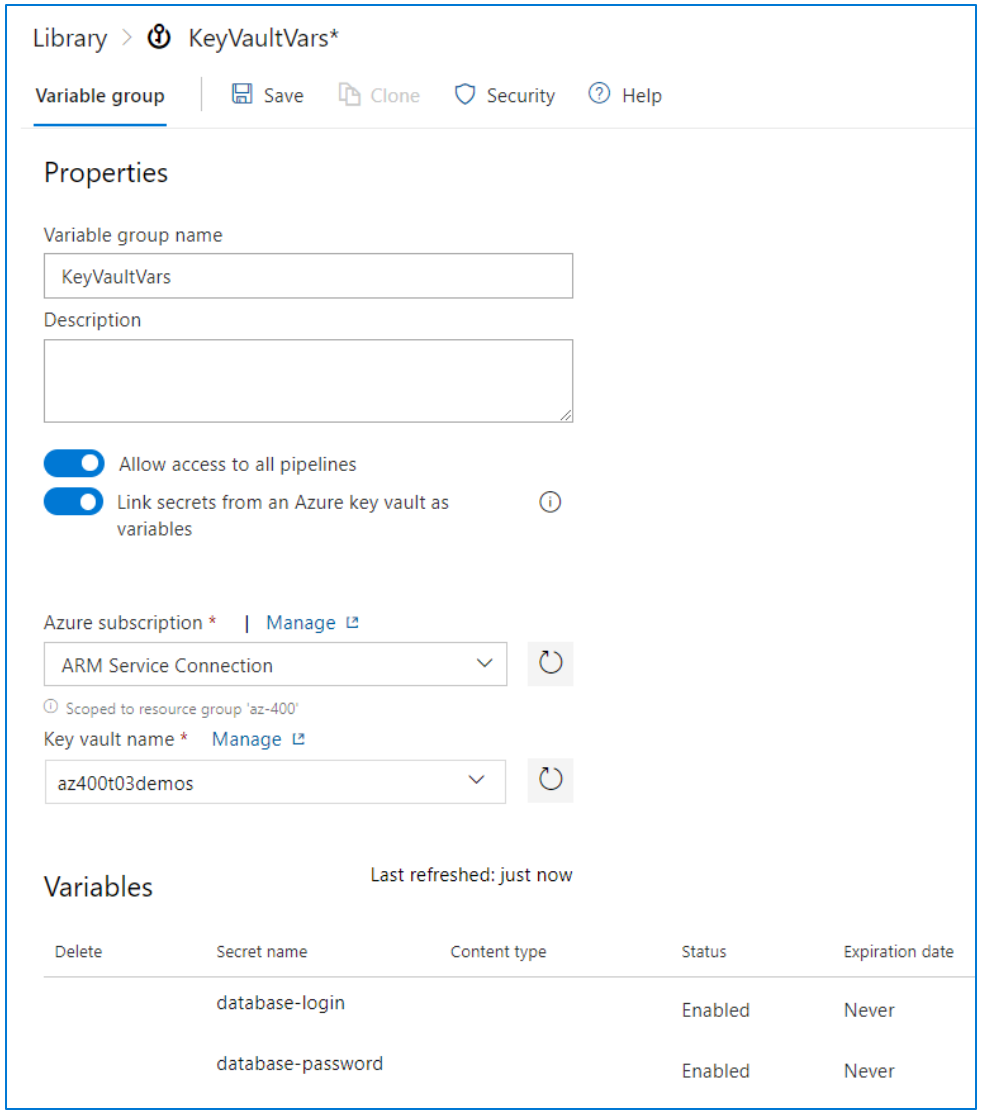
- Using Service Connections
- Using secret variables
- Storing secrets in a key vault

Using Key Vault in Pipelines

Secrets defined in Key Vault can be integrated using

- Key Vault Task -> Usage: $(database-login)
- Variable Groups

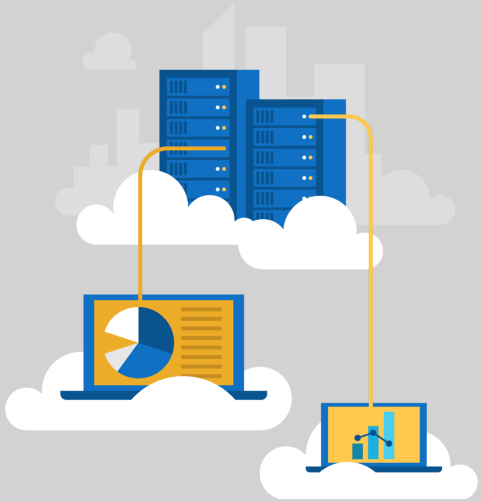




Setting up Secrets in the pipeline

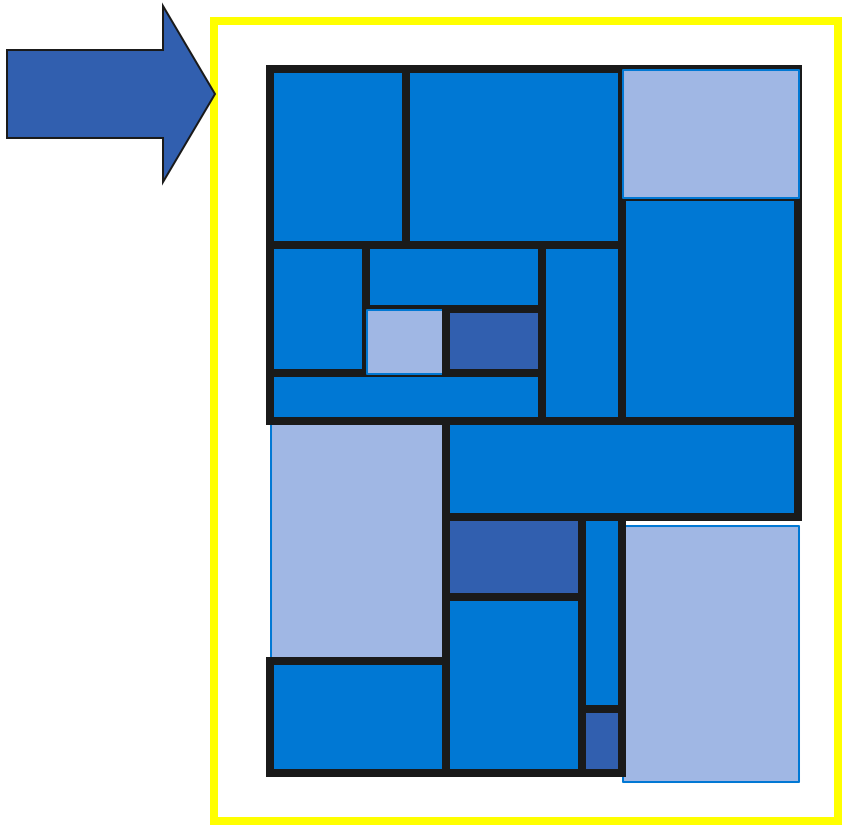
Setting up secrets in the pipeline with Azure Key vault

Lesson 05: Configure Automated Integration and Functional Test Automation



Traditional Testing

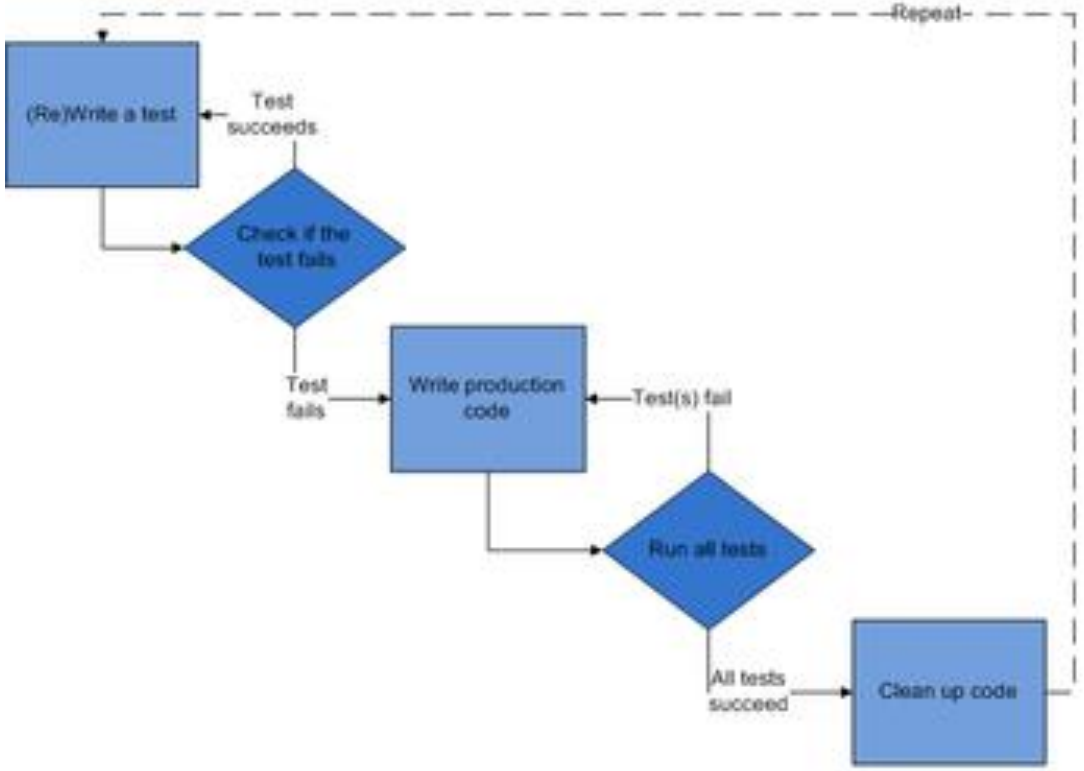
- Test the system as a whole
- Individual components rarely tested
- Errors go undetected
- Isolation of errors difficult to track down



Test-driven development (TDD)

For each new feature in the program:

1. Write (failing) test case
2. Run the test, to see that it fails
3. Write code until the test pass
4. Refactor the code to acceptable standards

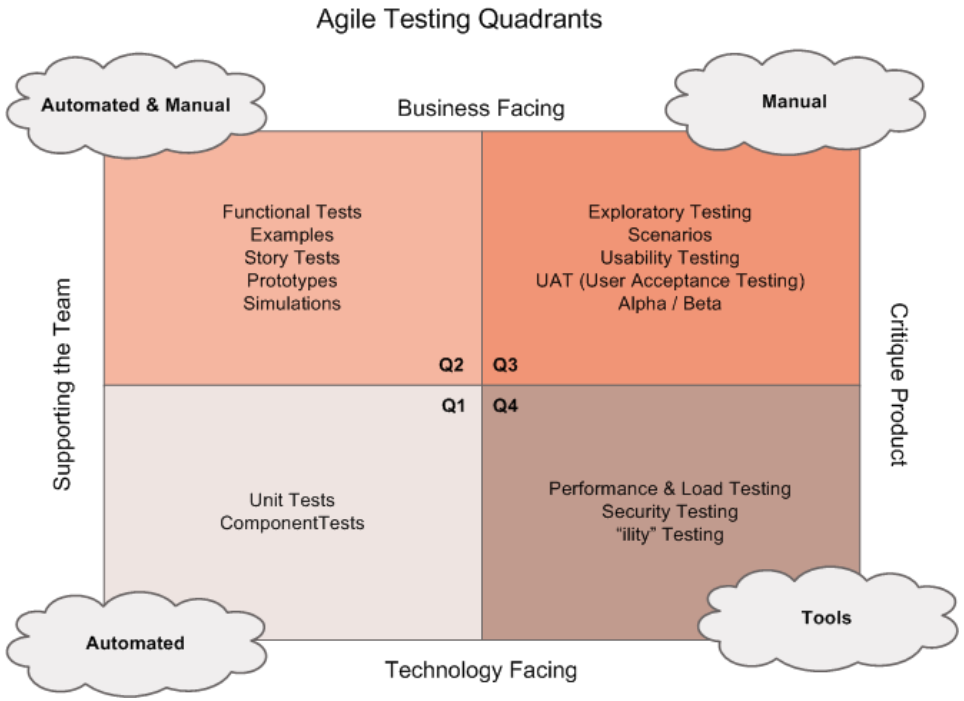


Test Types

- Smoke Testing
- Unit Testing
  - xUnit, Jest
- Integration Testing
- UI Testing
  - Selenium, Cypress
- Load Testing
  - Azure Test Plans

Configure Automated Integration and Functional Test Automation

- Do not automate all your manual tests
- Rethink test strategy
- Tests should be written at the lowest level possible
- Write once, run anywhere including production system
- Product is designed for testability
- Test code is product code, only reliable tests survive
- Test ownership follows product ownership



Source: https://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/

Setting up and Running Load Tests

Setting up and Running Functional Tests

Setting up and Running Availability Tests

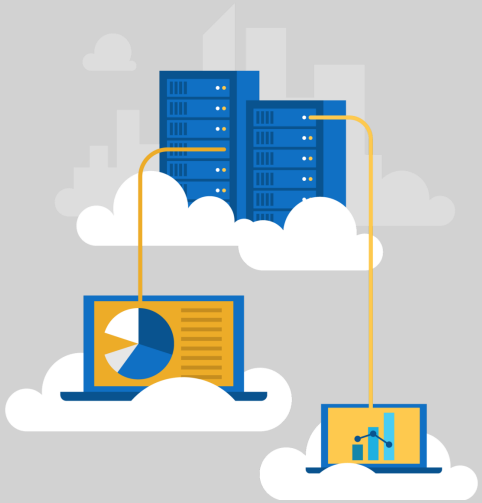
Create Health endpoints in your application

Use tools, e.g. availability tests in Application Insights, to test health endpoints

Two types of availability tests

- URL ping test
- Multi-step web test

Lesson 06: Automate Inspection of Health



Automate Inspection of Health

Helps you to stay informed about the state of your pipelines

Release Gates

Events, subscriptions, and notifications

Service Hooks

Reporting

Events, Subscriptions, and Notifications

Actions in Azure DevOps trigger events

Users can subscribe to events and get notified

- Calling a SOAP url
- Getting an email

Notifications can be managed centrally and personally

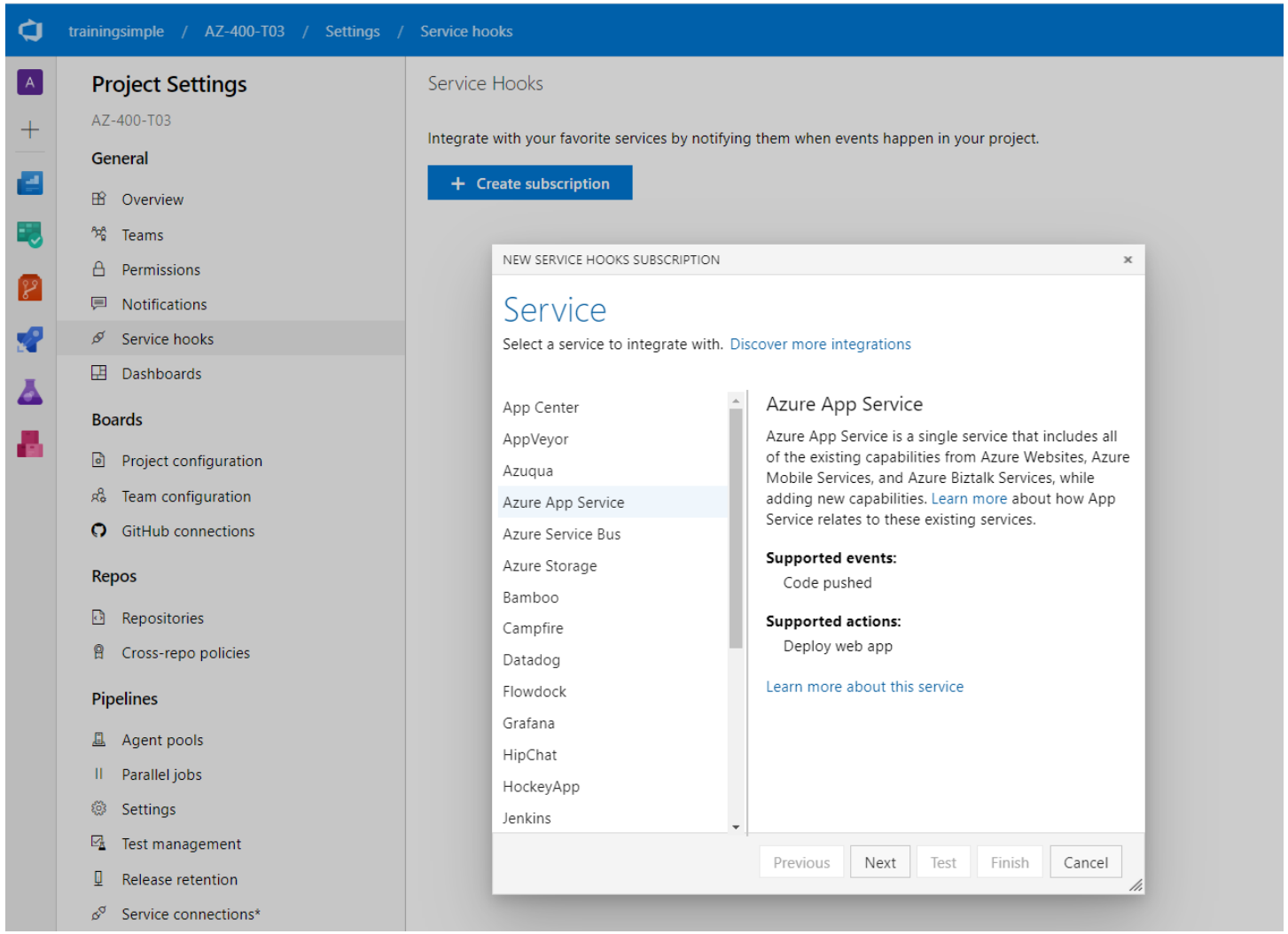
Service Hooks

- Service hooks enable you to perform tasks on other services when events happen
- Out of the Box integrations

Build and release	Collaborate	Customer support	Plan and track	Integrate
AppVeyor	Campfire	UserVoice	Trello	Azure Service Bus
Bamboo	Flowdock	Zendesk		Azure Storage
Jenkins	HipChat			Web Hooks
MyGet	Hubot			Zapier
Slack				

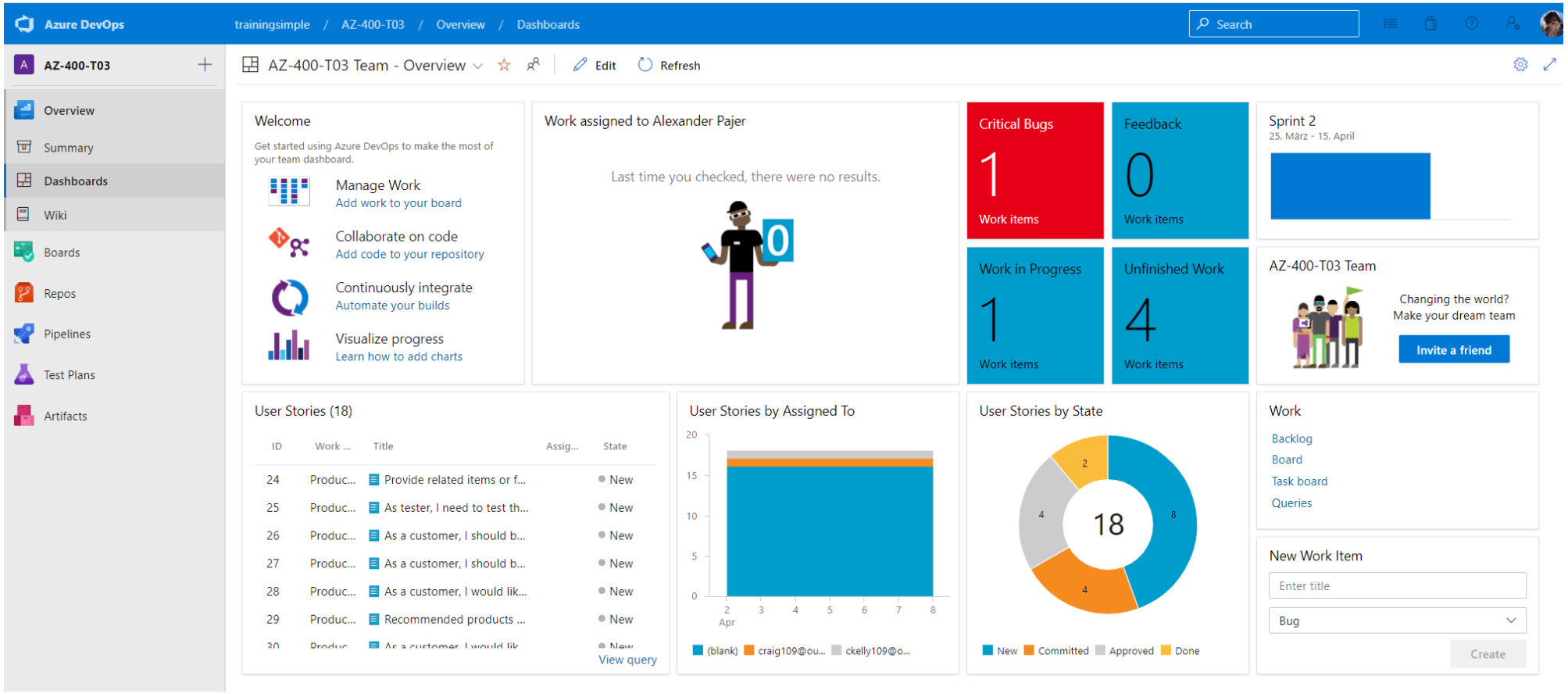
Create Service Hook

Done in Project Settings



Release Dashboards

Help you to gain visibility into your team's progress by adding one or more widgets or charts to your dashboard



Using Azure Monitor as Release Gate

Creating a Release Dashboard

Wrap Up

You are familiar with the terminology used in Azure DevOps and other Release Management Tooling

You know what a Build and Release task is, what it can do, and are familiar with some available deployment tasks

You know what an Agent, Agent Queue and Agent Pool is and what an agent is all about

You know what a release job is, and why you sometimes need multiple release jobs in one release pipeline

You know the difference between multi-agent and multi-configuration release job

You know what release variables are, what stage variables are and how to best use them in your release pipeline

You know what a Service Connection is and how you can use them to deploy to an environment securely

You know what the possibilities for testing are in the pipeline and how to embed testing in the pipeline

You are familiar with the different ways to inspect the health of your pipeline and release by using, alerts, service hooks and reports

You know what a release gate is and how to create one

Module 2: Review Questions

1. How many deployment jobs can be run concurrently by a single agent?
2. What should you create to store values that you want to make available across multiple build and release pipelines?
3. How can you provision the agents for deployment groups in each of your VMs?

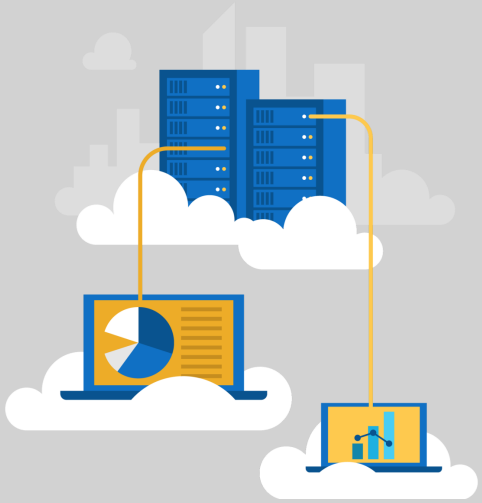
Microsoft

AZ-400T03
Module 3: Implement an Appropriate Deployment Pattern

Learning Objectives

- You learned that architecture is a prerequisite for achieving Continuous Delivery
- You know different Deployment Patterns
- You know what Blue-Green Deployment is, and how to use deployment slots in Azure to implement this
- You know what a canary release is, and how to use Azure Traffic Manager to implement this
- You know what a feature toggle is, and how they are of great importance when implementing Continuous Delivery
- You know what a Dark Launch is and how this differs from a canary release
- You know the concept of A/B testing and how this is enabled by Continuous Delivery
- You know what Progressive Exposure Deployment and Ring based deployments are and how to implement this in Azure DevOps

Lesson 01: Introduction into Deployment Patterns



Introduction

Continuous Delivery is more than Release Management

Deployment is only a step of the Release Cycle -> Requirements:

- Testing Strategy
- Good Coding Practices, Safe Coding
- Architecture

Monoliths are hard to deliver because of all the dependencies

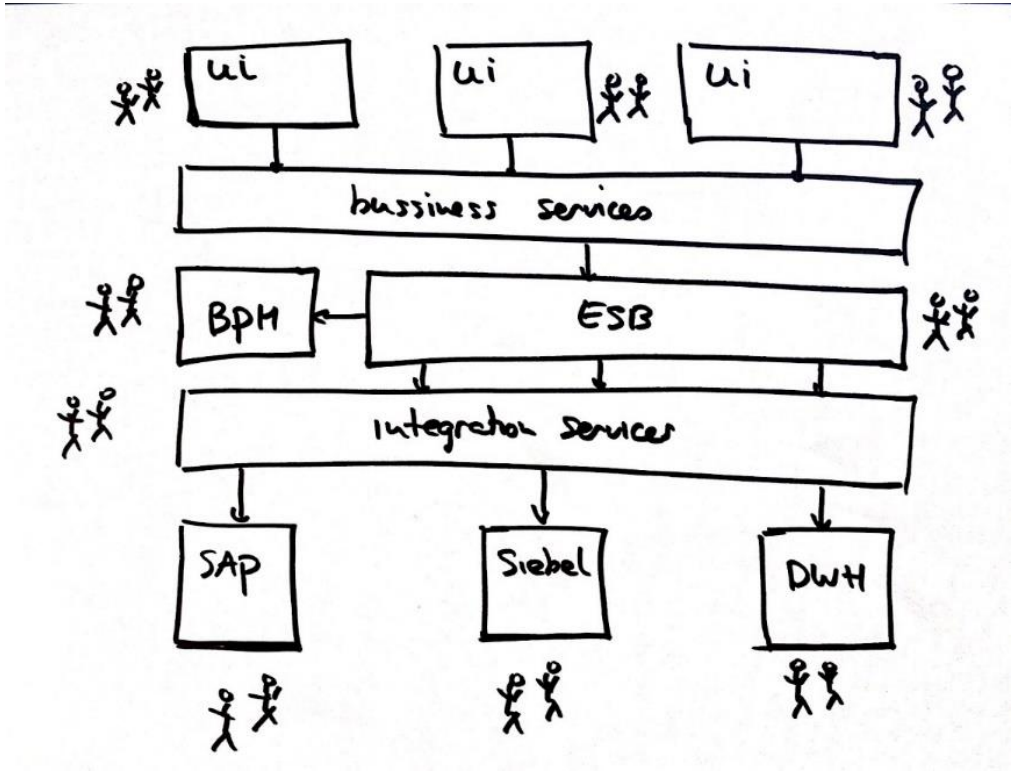
- Break up in smaller pieces
- Microservices

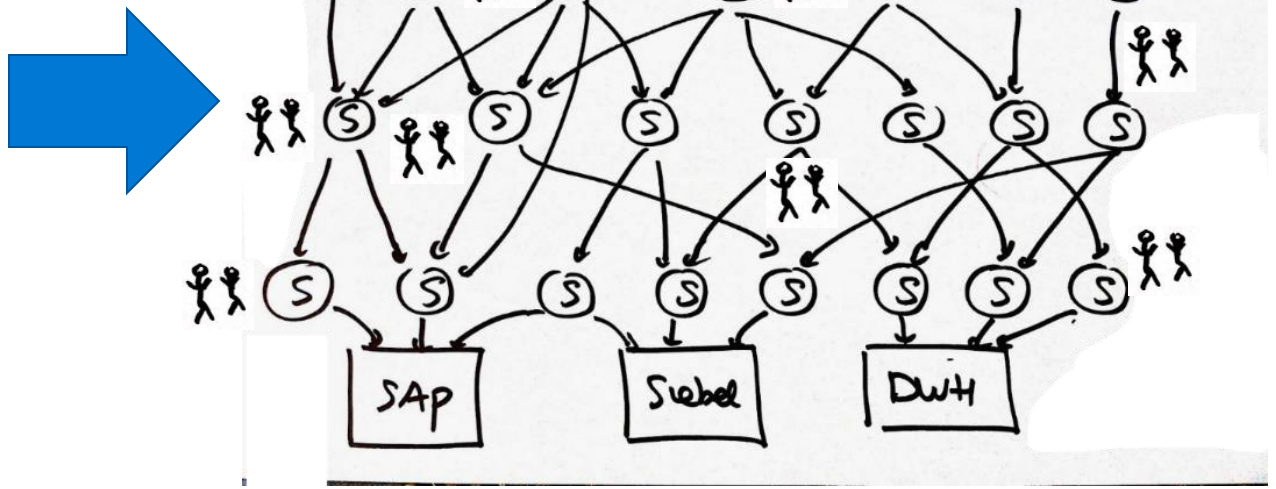
Definitions of Microservices

There is no single definition for microservices. A consensus view has evolved over time in the industry.

- Services in a microservice architecture (MSA) are often processes that communicate over a network to fulfil a goal using technology-agnostic protocols such as HTTP.[2][3][4]
- Services in a microservice architecture are independently deployable.[5][6]
- Services are organized around business capabilities.[7]
- Services can be implemented using different programming languages, databases, hardware and software environment, depending on what fits best.[6]
- Services are small in size, messaging-enabled, bounded by contexts, autonomously developed, independently deployable, decentralized and built and released with automated processes.[5]

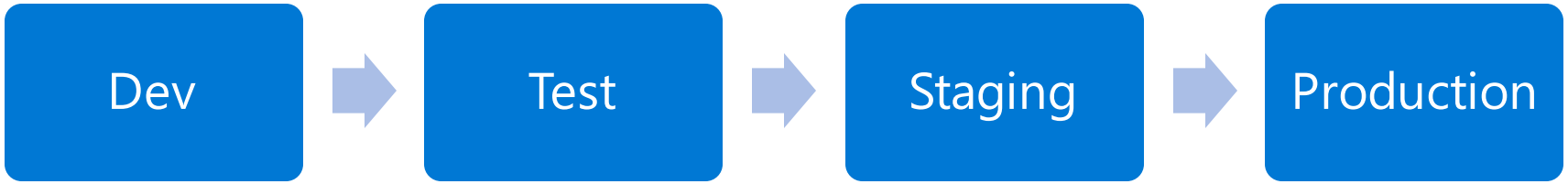
Microservices architecture





Deployment Patterns

“Classical” Deployment Patterns

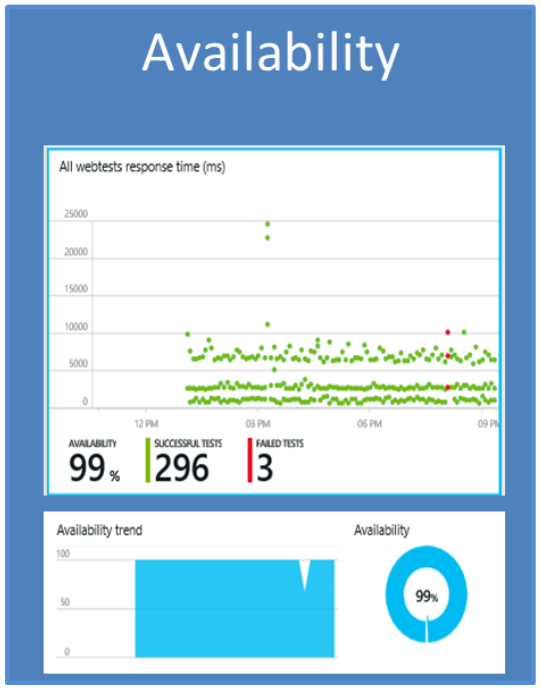


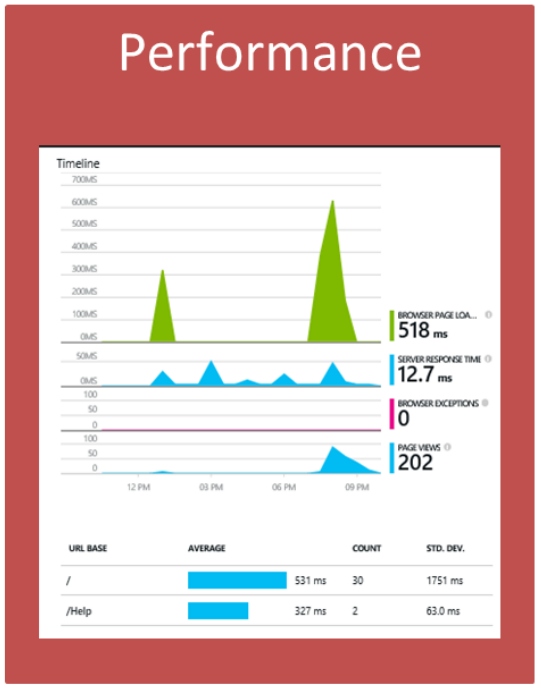
Modern Practices:

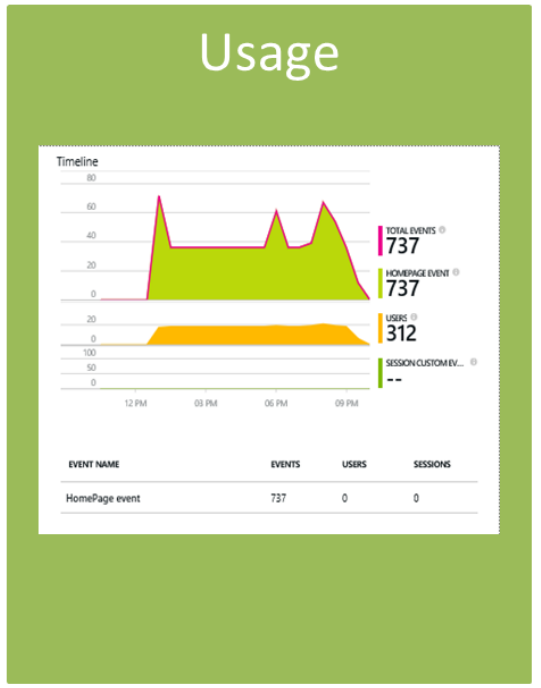
- Blue Green Deployments
- Canary Releases
- Dark Launching
- A/B Testing
- Progressive Exposure Deployment / Ring Based Deployments

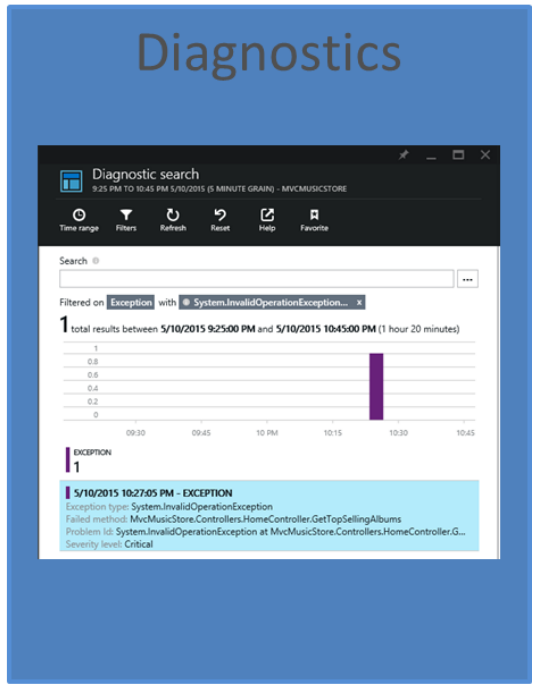
Feedback loops

E.g. use application insights







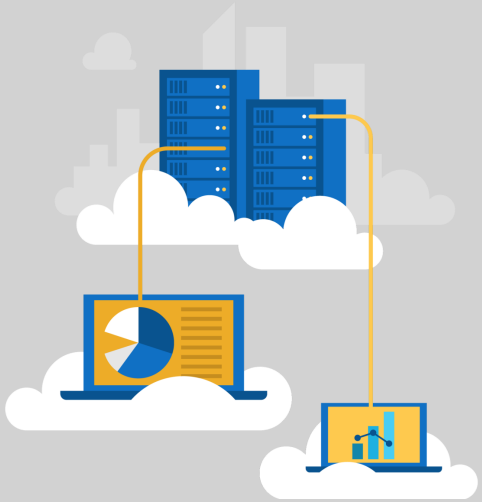


Considerations for cadence

- Do we want/need to deploy every day?
- What is your target environment?
- Is it used by one team or is it used by multiple teams?
- Who are the users? Do they want a new version multiple times a day?
- How long does it take to deploy?
- Is there downtime? What happens to performance? Are users impacted?

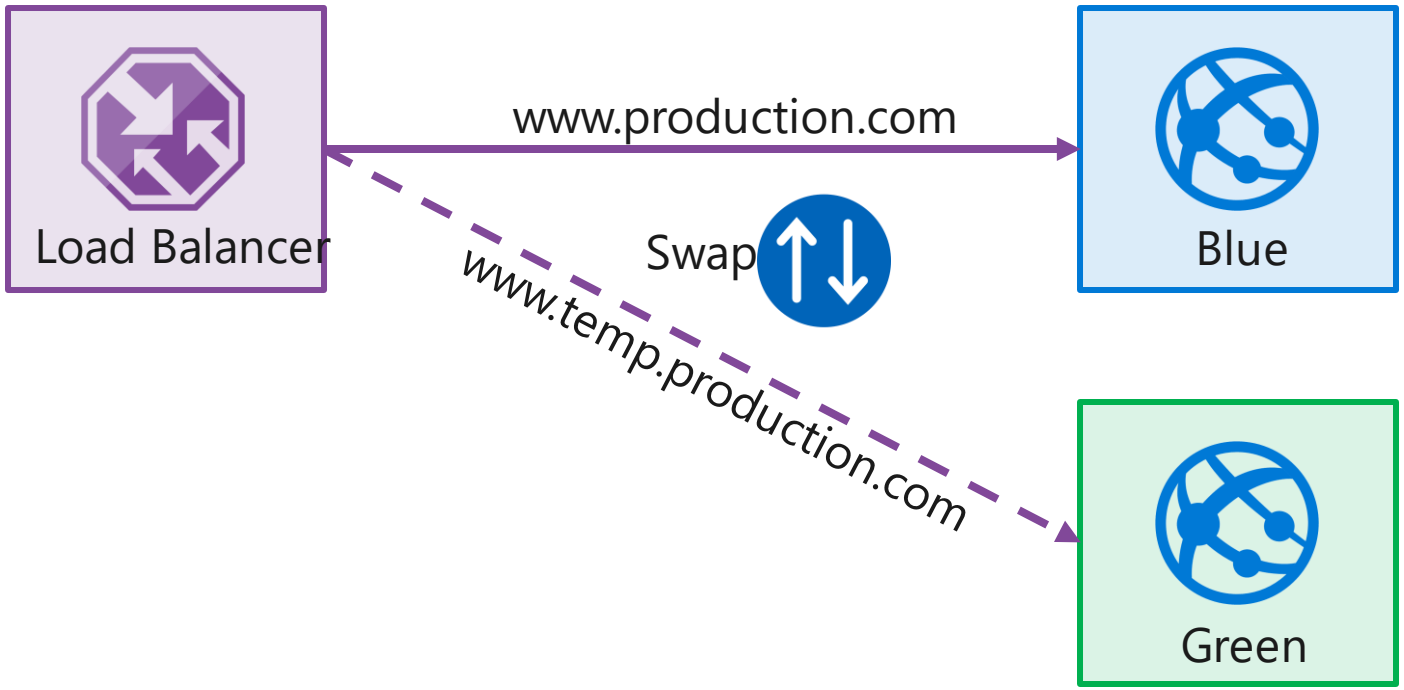
To be in cadence (... with the bussiness needs) - Im Einklang sein

Lesson 02: Implement Blue Green Deployment



Blue Green deployment

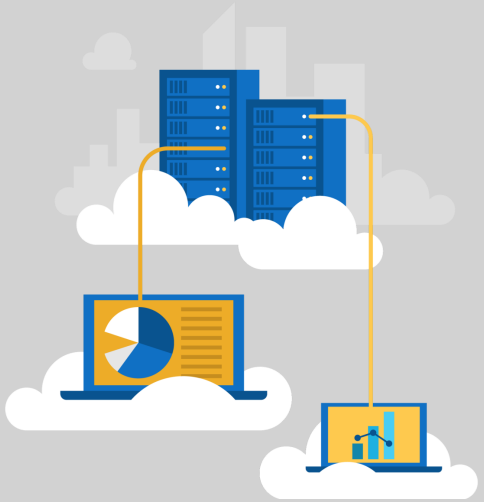
Enabled using Deployment Slots



Setting up a Blue-Green deployment

Blue-Green Deployment

Lesson 03: Feature Toggles



Feature toggles | flag

What is it?

Mechanism to separate feature deployment from feature exposure

A.k.a. feature flippers, feature flags, feature switch, conditional feature, etc.

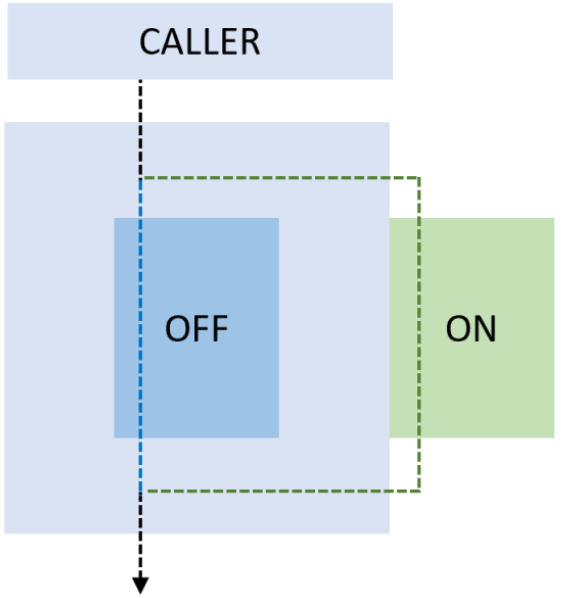
Why do you need it?

It enables you to give control back to the business on when to release the feature

Enables A/B testing, canary releases and dark launching

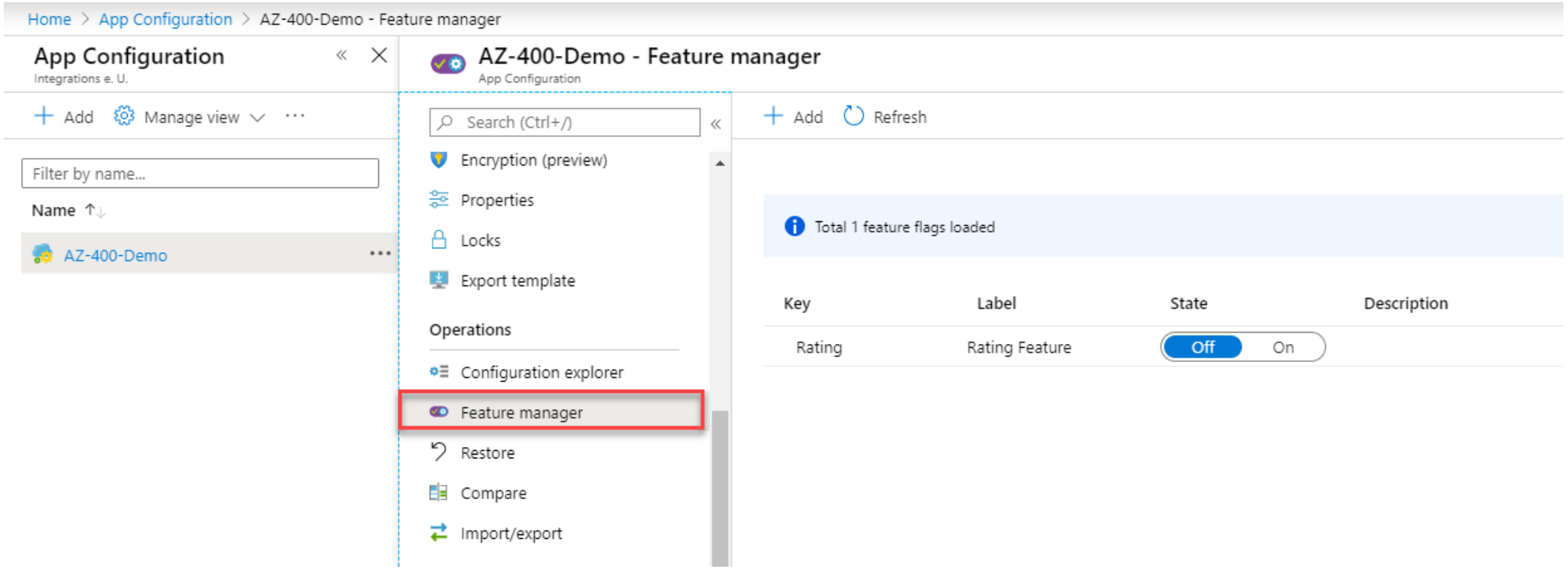
It provides an alternative to keeping multiple branches in version control

Enables change without redeployment

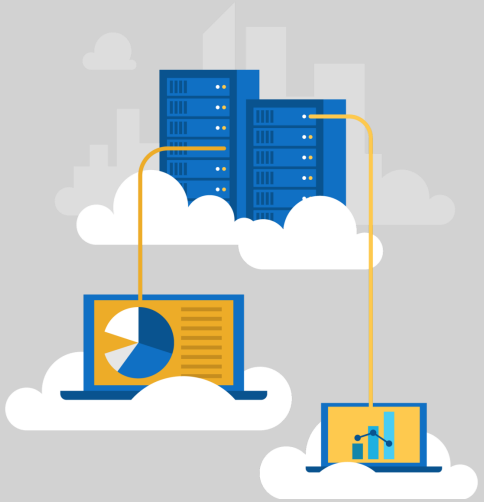


Implementing & Managing Feature Toggles

Could be implemented manually or using Azure App Configuration



Lesson 04: Canary Releases



Canary release

What is it?

- Releasing a feature to a limited subset of end users

Why do you need it?

- Validates there is no break
- When you want to gradually roll out
- a feature to ensure enough capacity

Canary release

How do you do it?

- Use a combination of feature toggles, traffic routing and deployment slots
- Random selection of users:
  - Setup deployment slots with feature enable
  - Route %traffic to the instance with the new feature enabled
  - Target specific user segment
- Use feature toggle to enable feature for specific user segment

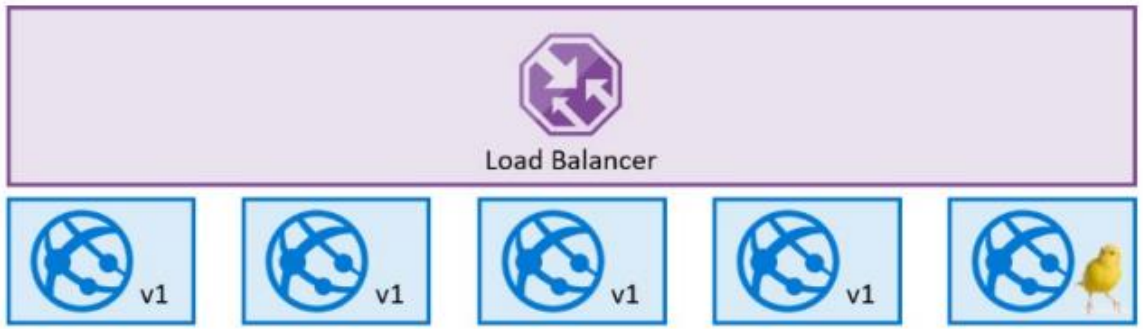
Traffic Manager

What is it?

- Traffic manager provides the ability to route traffic between Azure app services
- Within an app service, routes traffic between deployment slots

Why do you need it?

- It enables failover and load distribution capabilities
- It enables you to deploy to a slot and then slowly move traffic over to the other slot



Traffic manager options

Traffic manager provides four options:

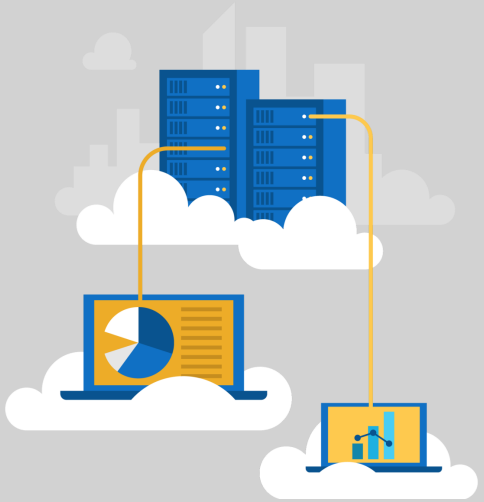
1. Route traffic based on availability
2. Route traffic round robin
3. Route traffic based on weight
4. Route traffic based on latency
5. Geography

In the context of continuous delivery, we are most interested in option 1 and 3

Setting up a Ring Based Deployment with Traffic Manager

Traffic Manager

Lesson 05: Dark Launching



Dark Launching

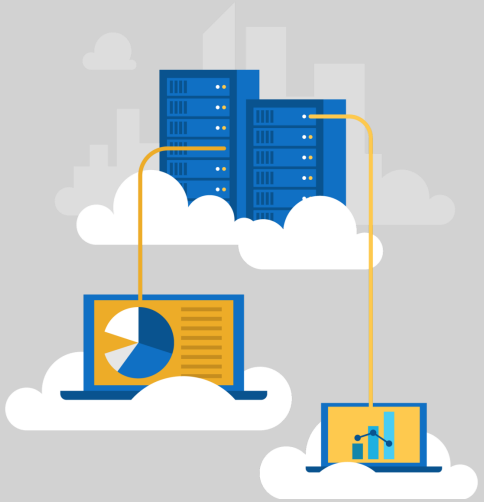
Dark Launching is in many ways similar to Canary Testing.

However, the difference here is that you are looking to assess the response of users to new features in your frontend, rather than testing the performance of the backend.

The idea is that rather than launch a new feature for all users, you instead release it to a small set of users.

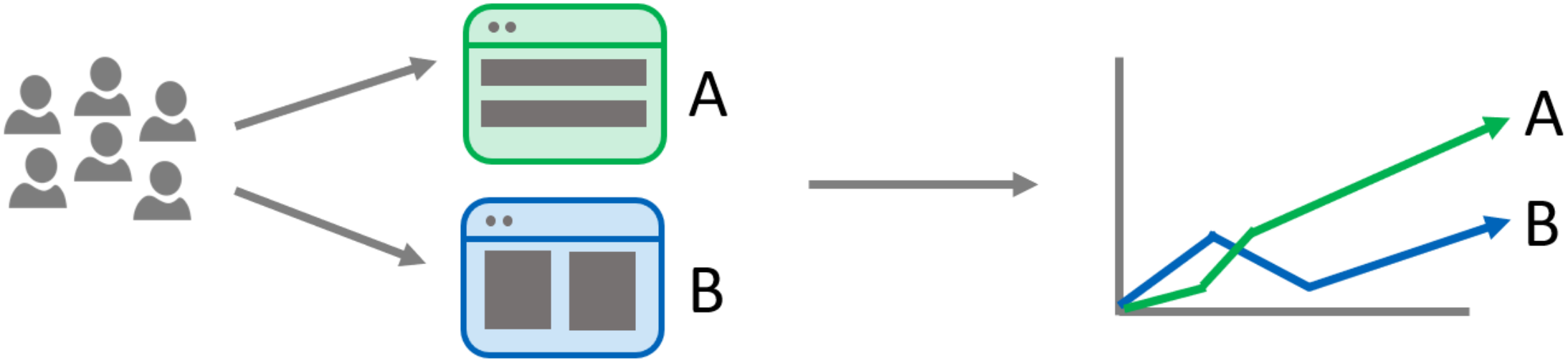
Usually, these users aren't aware they are being used as guinea pigs for the new feature and often you don't even highlight the new feature to them, hence the term "Dark" launching.

Lesson 06: AB Testing



A/B Testing

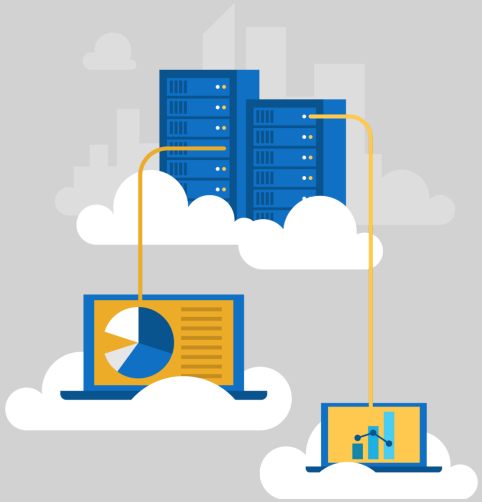
A/B testing is mostly an experiment where two or more variants of a page are shown to users at random, and statistical analysis is used to determine which variation performs better for a given conversion goal.



Why A/B Testing

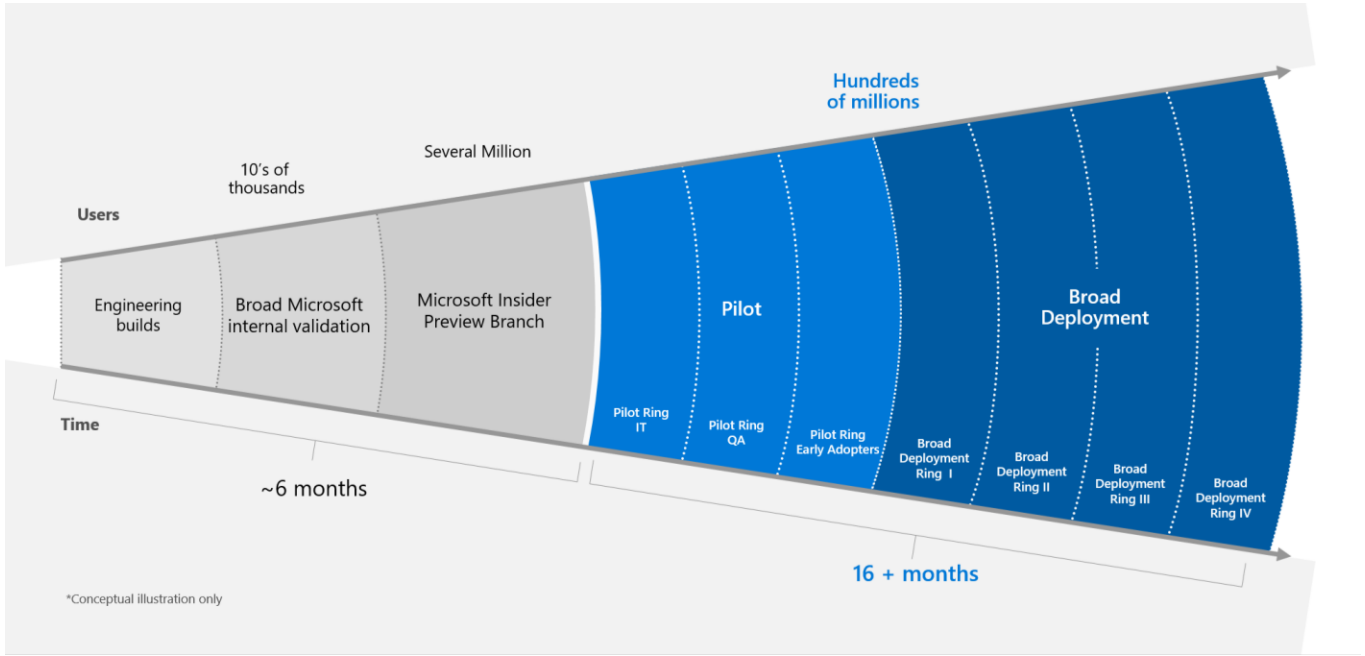
- Experiment on features and usage
- Improve conversion rate
- Continuous experiments
- Measure impact of change

Lesson 07: Progressive Exposure Deployment

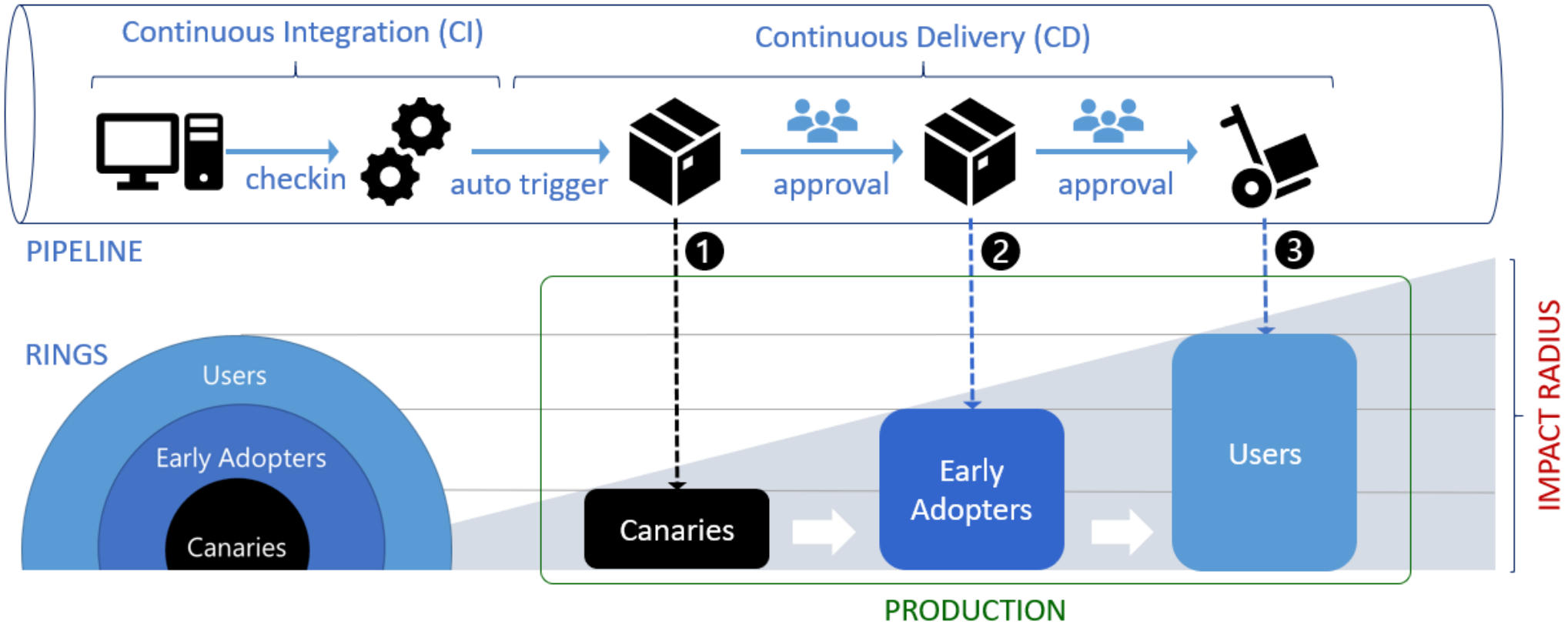


What is Progressive Exposure Deployment

- A.k.a. Ring Based Deployment
- Deploy changes to risk tolerant customers first, progressively roll out to larger and larger sets of customers
- Automated health checks
- Possibility to stop or roll back



CI/CD with deployment rings



Deployment Definition

In Azure DevOps, rings are modeled as Stages