

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

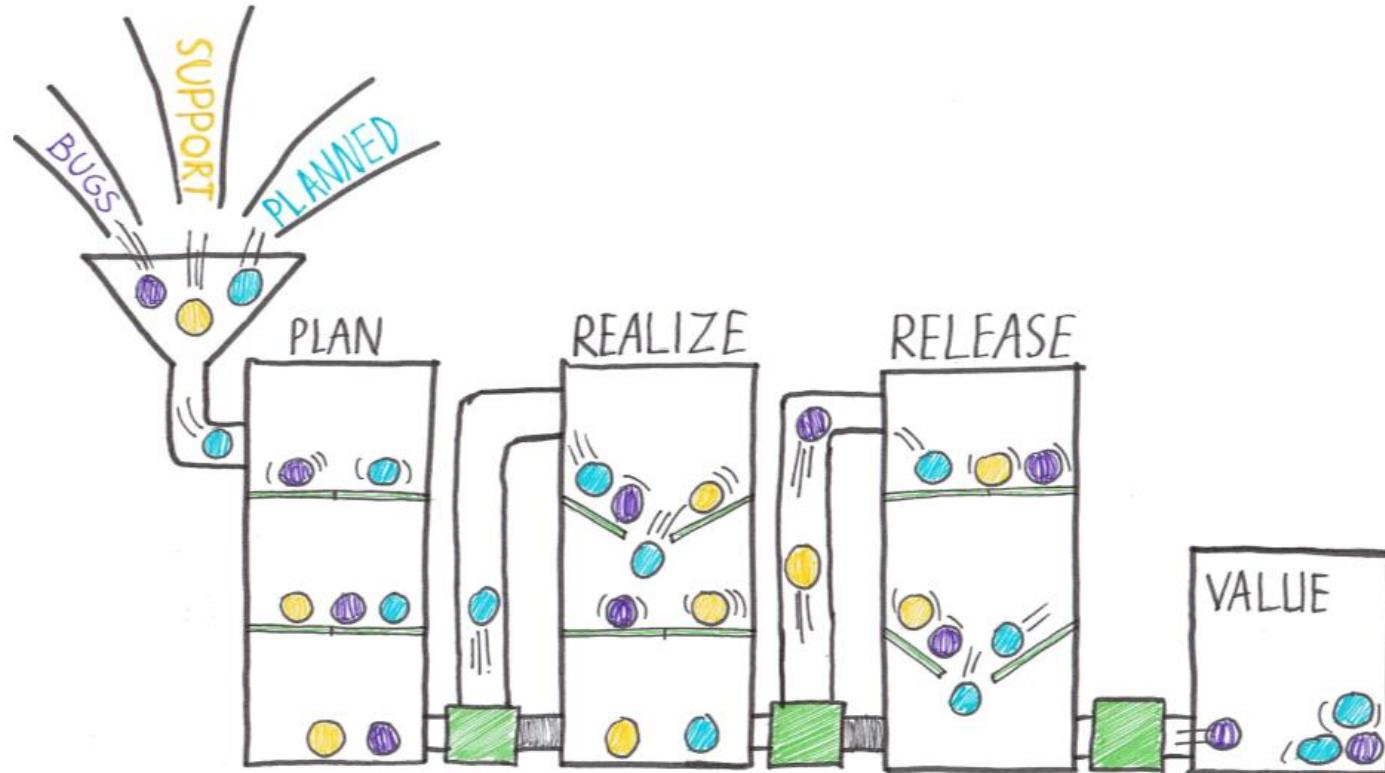
Why is this the case?

- 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

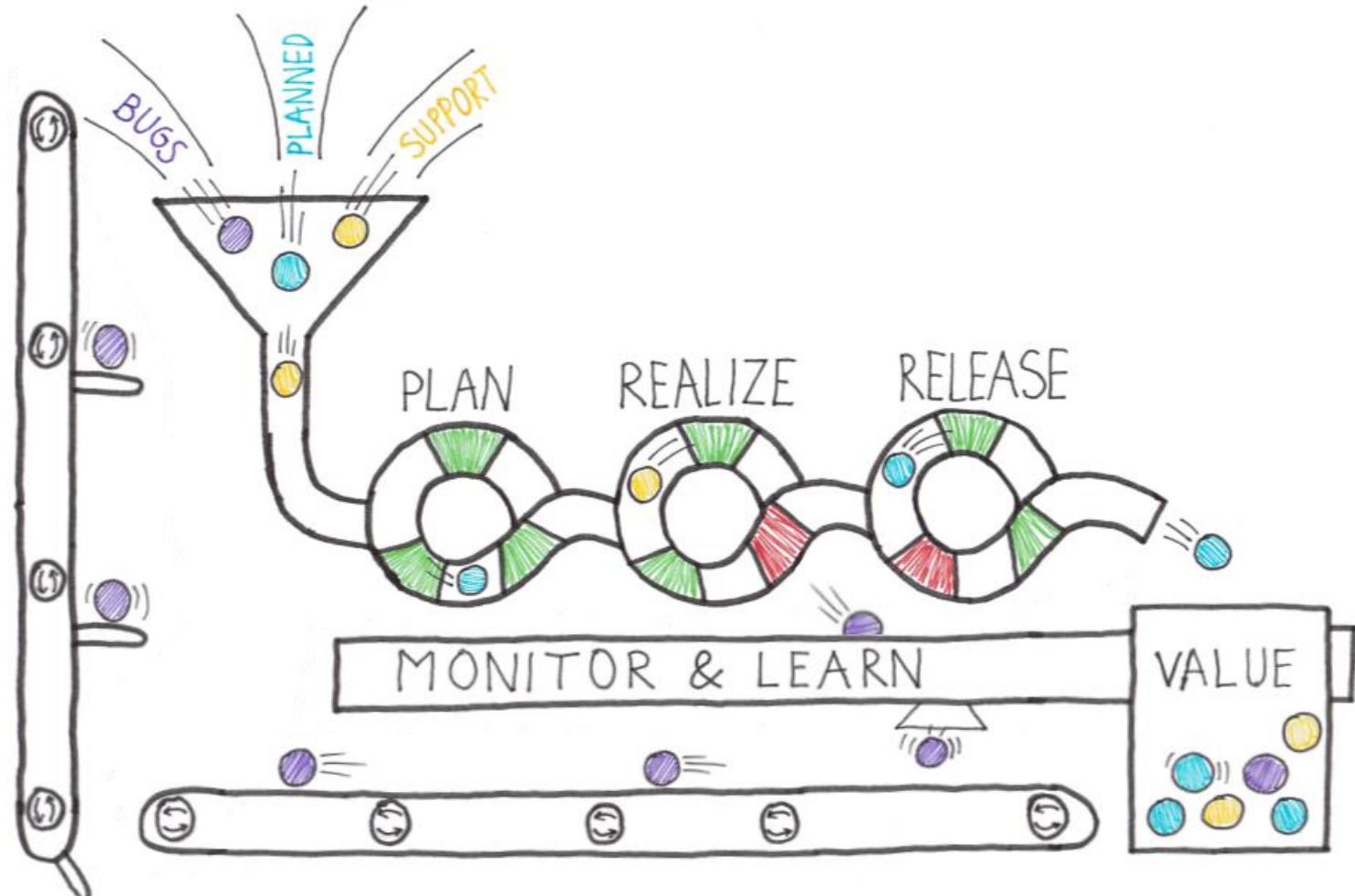
Cheaper

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

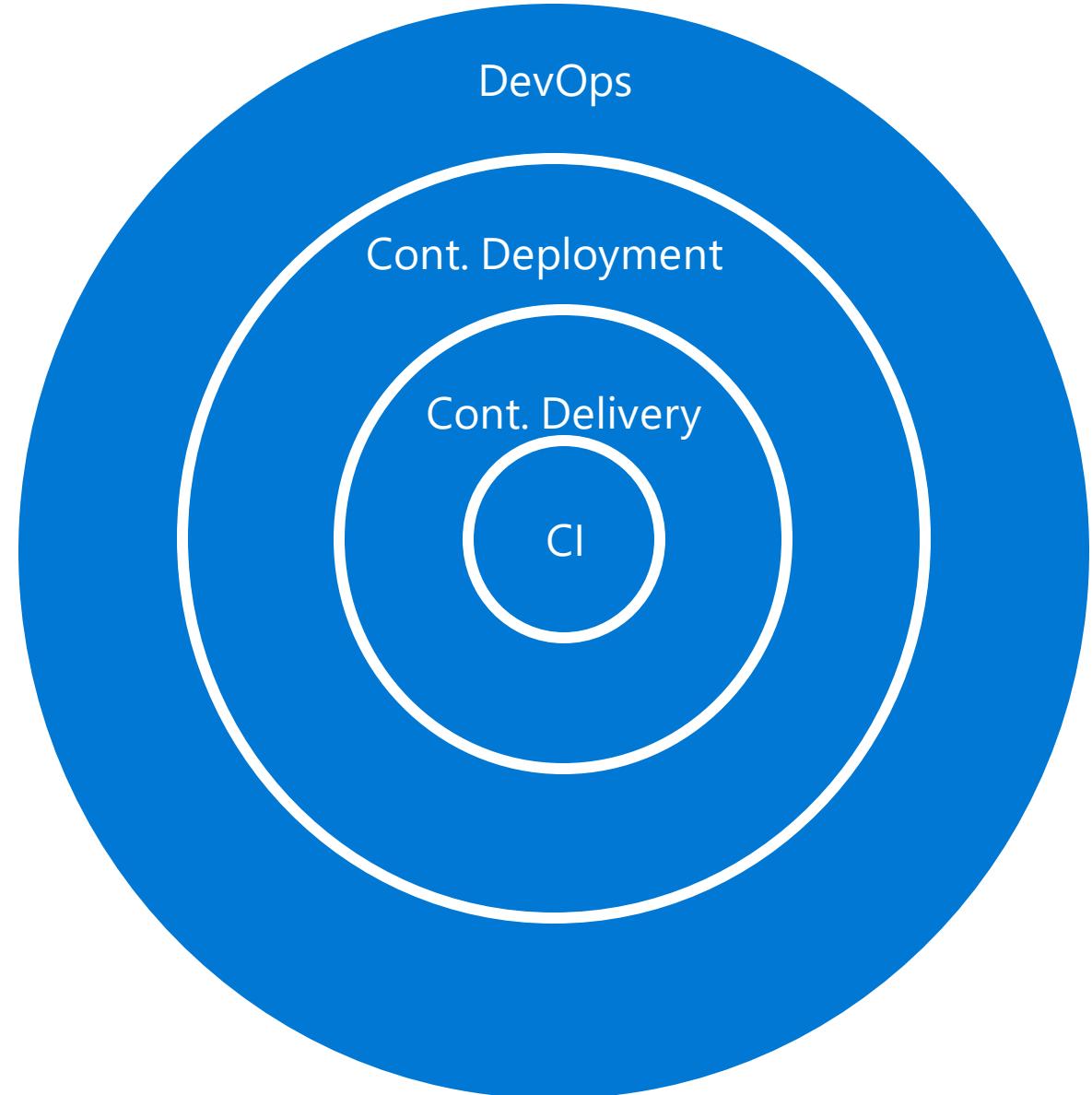
Continuous Delivery

- 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

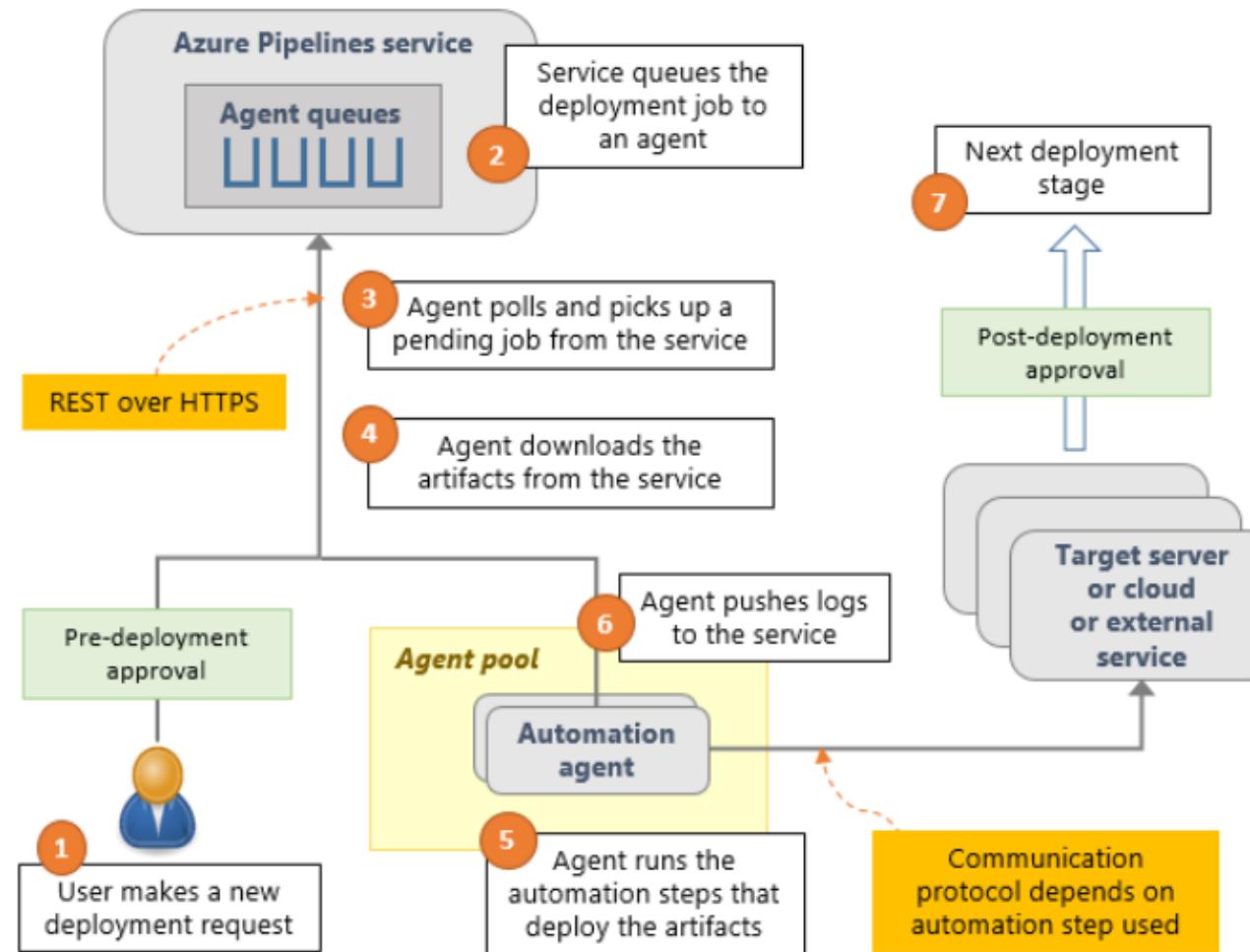
- Release and deployment are often coupled
- Release is not the same as a deployment
- Separate that in functional and technical release
 - Functional release is exposing features to customers
 - Technical release is deploying functionality

Release vs Deployment

Use similar tools but differentiate by:

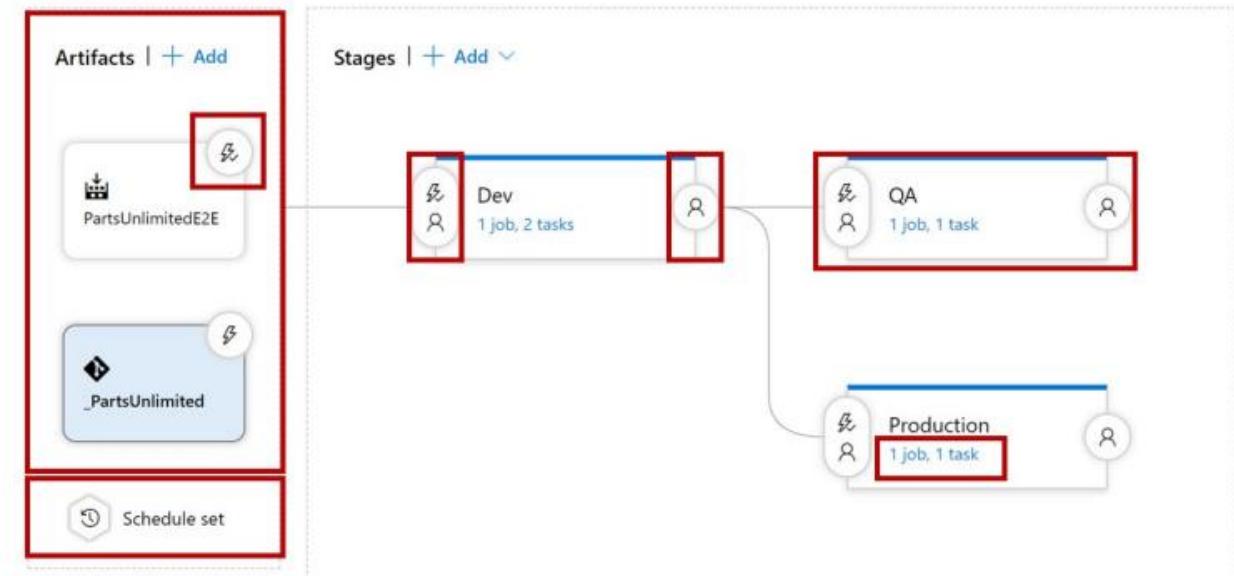
- Release:
 - Collection of versioned artifacts, pipeline, approvals, stages, variables
 - When authoring a release pipeline, you link the appropriate artifact sources to your release pipeline
- Deployment:
 - The Release provisioned to a specific infrastructure environment
 - Infrastructure can also 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.
- Organization:
 - 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.)

Create new feed ×

Feeds host your packages and let you control permissions.

Name This name will appear in the URL for your feed

Project: Packages
New: The feed will be scoped to the Packages project. [Learn more about project-scoped feeds](#). If this project is made public, its scoped feeds will also become public.

Visibility

Members of your Azure Active Directory
Any member of your AAD can view the packages in this feed

Members of integrationstraining
Any member of your organization can view the packages in this feed

Specific people
Only users you grant access to can view the packages in this feed

Upstream sources

Include packages from common public sources
For example: nuget.org, npmjs.com

Feed Permissions

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

Tailspin.SpaceGame.Web.Models > Feed settings

User/Group	Role	Inherited
Alexander Pajer	Owner	
[Packages]\Project Administrators	Owner	
[integrationtraining]\Project Collection Administrators	Owner	
Project Collection Build Service (integrationtraining)	Contributor	
[Packages]\Contributors	Contributor	

The row for "Project Collection Build Service (integrationtraining)" is highlighted with a red box.

The screenshot shows the Azure DevOps interface for managing feeds. The top navigation bar includes 'Feed details', 'Permissions' (which is selected), 'Views', 'Upstream sources', and buttons for 'Add users/groups', 'Delete', and '...'. Below this is a table of permissions. To the right, the main area displays the feed settings for 'Tailspin.SpaceGame...' under 'integrationtraining / Packages / Artifacts / Packages'. It shows an 'Overview' section with links for 'Boards', 'Repos', 'Pipelines', 'Test Plans', and 'Artifacts'. A message says 'You can now search for packages across feeds. Try searching for a package in the search'. Below this is a 'Filter by keywords' input field. A table lists packages with columns for 'Package', 'Views', and 'Last pushed'. One entry is shown: 'Tailspin.SpaceGame.Web.Models Version 1.0.0-CI-20200324-181902 Just now'.

Connect to Feed VS Code

The screenshot shows the Azure Artifacts interface for connecting a Visual Studio Code extension to a package feed. On the left, a sidebar lists various feeds: NuGet, dotnet, NuGet.exe, Visual Studio, npm, npm, Maven, Maven, Gradle, Python, pip, and twine. The 'dotnet' feed is selected, indicated by a highlighted row.

dotnet Command line reference

First time using Azure Artifacts with dotnet on this machine? [Get the tools](#) [X](#)

Project setup

Add a nuget.config file to your project, in the same folder as your .csproj or .sln file

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="Tailspin.SpaceGame.Web.Models" value="https://pkgs.dev.azure.com/integrationtraining/AZ-400-T03/_packaging/Tailspin.SpaceGame.Web.Models/nuget/v3/index.json" />
  </packageSources>
</configuration>
```

Restore packages

Restore packages (using the interactive flag, which allows dotnet to prompt you for credentials)

```
dotnet restore --interactive
```

Note: You don't need --interactive every time, dotnet will prompt you to add --interactive if it needs updated credentials.

Publish packages

Publish a package by providing the package path, an API Key (any string will do), and the feed URL

```
dotnet nuget push --source "Tailspin.SpaceGame.Web.Models" --api-key az <package-path>
```

Connect to Feed VS Prof

The screenshot shows the Visual Studio NuGet Package Manager interface. On the left, a sidebar lists various package sources: NuGet, dotnet, NuGet.exe, Visual Studio, npm, npm, Maven, and Maven. The 'Visual Studio' section is active, showing a 'Machine setup' dialog. This dialog instructs the user to select Options > NuGet Package Manager > Package Sources in the Tools menu. It displays a 'Name' field set to 'Tailspin.SpaceGame.Web.Models' and a 'Source' field set to 'https://pkgs.dev.azure.com/integrationtraining/Packages/_packaging/Tailspin...'. A note at the bottom says: 'Note: You need to do this on every machine that needs access to your packages. Use the NuGet.exe instruction to add your repository.' At the top of the main area, there's a message: 'You can now search for packages across feeds. Try searching for a package in the search box. Learn more.' To the right of the message is a 'Get the tools' button. The main content area has a title 'NuGet Package Manager: Tailspin.SpaceGame.Web'. In the top right corner of this area, a 'Package source: nuget.org' dropdown is highlighted with a red box. Below this, a large modal window titled 'Options' is open. The 'Available package sources:' section contains two entries: 'nuget.org' (checked) and 'Tailspin.SpaceGame.Web.Models' (checked). The 'Name' field in the 'Machine-wide package sources:' section is also highlighted with a red box and set to 'Tailspin.SpaceGame.Web.Models'. The 'Source' field is set to 'Tailspin.SpaceGame.Web.Models/nuget/v3/index.json'. At the bottom right of the modal are 'OK' and 'Cancel' buttons.

You can now search for packages across feeds. Try searching for a package in the search box. [Learn more](#).

← Connect to feed

Visual Studio

Learn about restoring packages

Get the tools

First time using Azure Artifacts with Visual Studio on this machine?

NuGet

dotnet

NuGet.exe

Visual Studio

npm

npm

Maven

Maven

Machine setup

Name: Tailspin.SpaceGame.Web.Models

Source: https://pkgs.dev.azure.com/integrationtraining/Packages/_packaging/Tailspin...
Note: You need to do this on every machine that needs access to your packages. Use the NuGet.exe instruction to add your repository.

Get 1

Include prerelease

NuGet Package Manager: Tailspin.SpaceGame.Web

Package source: nuget.org

Options

Search Options (Ctrl+E)

Available package sources:

- nuget.org
https://api.nuget.org/v3/index.json
- Tailspin.SpaceGame.Web.Models
https://pkgs.dev.azure.com/integrationtraining/Packages/_packaging/Tailsp...

Machine-wide package sources:

- Microsoft Visual Studio Offline Packages
C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

Name: Tailspin.SpaceGame.Web.Models

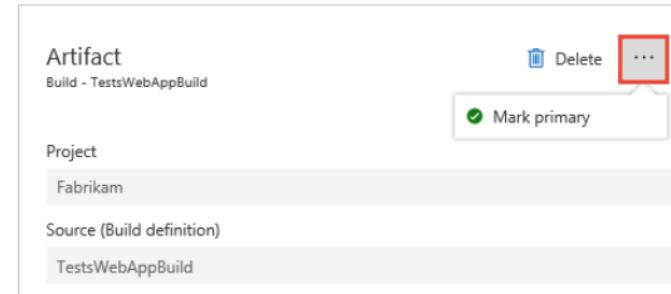
Source: Tailspin.SpaceGame.Web.Models/nuget/v3/index.json

Update

OK Cancel

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 chosen 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, ...)

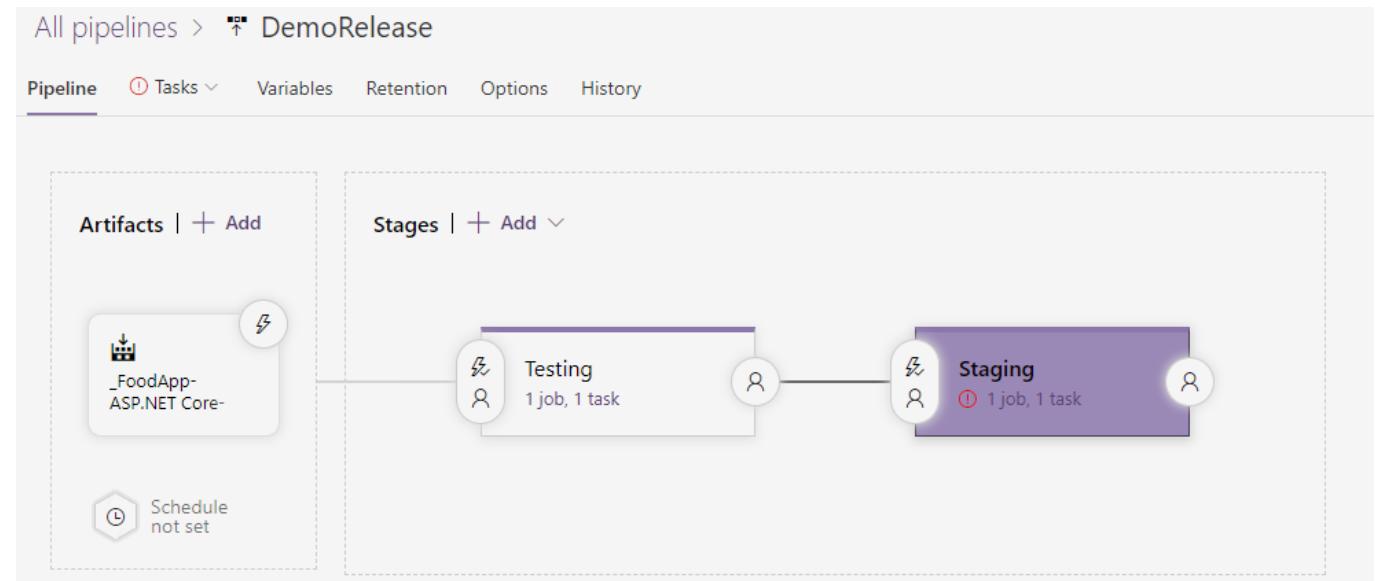
Can be short or long lived

What is your target environment?

Long-lived or short-lived environments

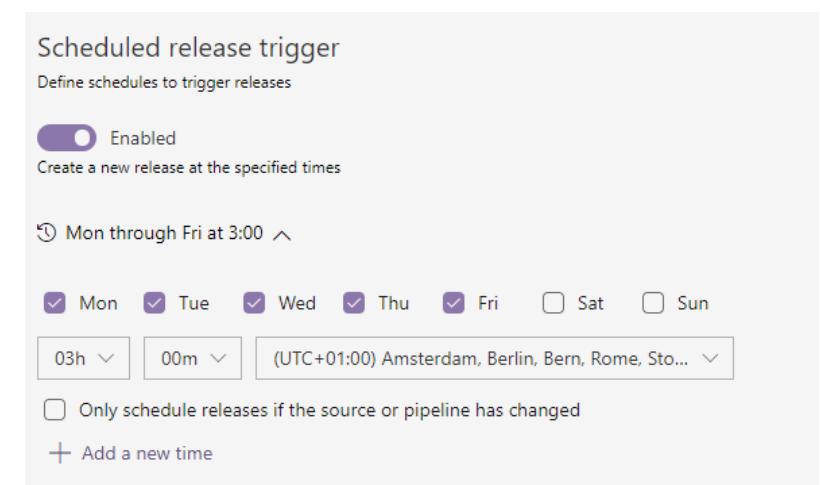
Purpose of an environment

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



The screenshot shows the Azure DevOps Pipeline editor for the 'DemoRelease' pipeline. On the left, the pipeline structure is visible with stages: 'Testing' and 'Staging'. On the right, two trigger configurations are listed: 'Continuous deployment trigger' and 'Pull request trigger'. Both triggers are currently disabled. The 'Continuous deployment trigger' is associated with a build named '_FoodApp-ASP.NET Core-T02-Demo01'. The 'Pull request trigger' is also associated with the same build. A red box highlights the trigger icon in the pipeline stages area.

Selecting an artifact source



Setting up deployment stages

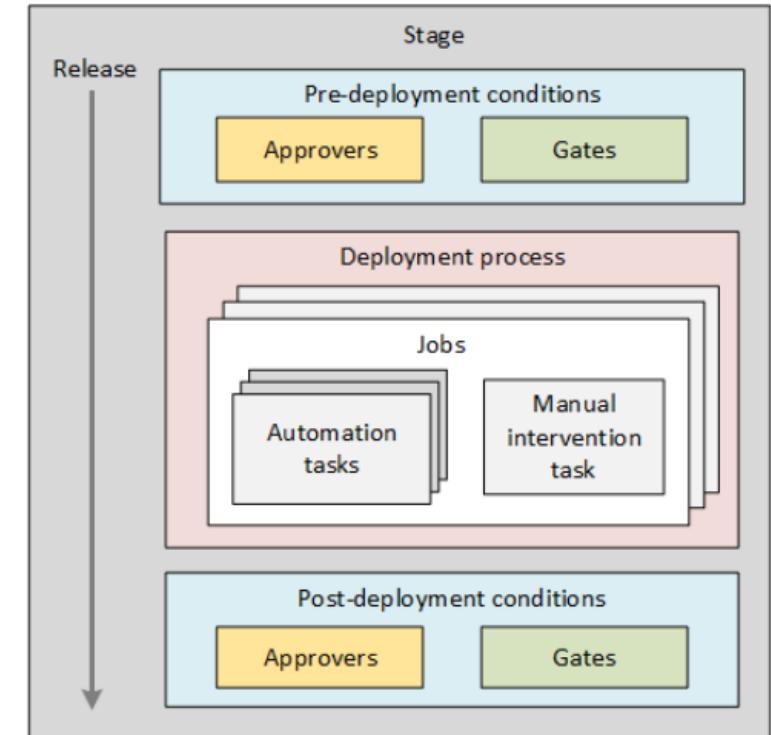


Lesson 03: Building a High Quality Release Pipeline



Controlling Release

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



Release Approvals

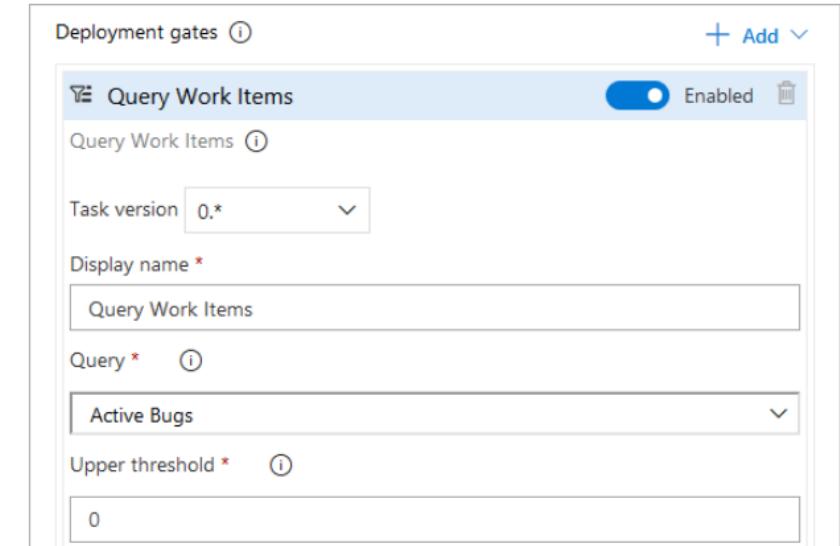
- Considerations for manual approvals
 - What do we want to achieve with the approval?
 - Who needs to approve?
 - When do you want to approve?

Release Gates

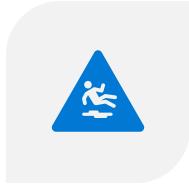
Give you additional control over the start and completion of the deployment pipeline.

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



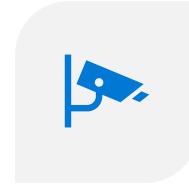
INCIDENT AND ISSUES
MANAGEMENT



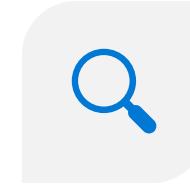
NOTIFICATION OF
USERS BY
INTEGRATION WITH
COLLABORATION
SYSTEMS



QUALITY VALIDATION



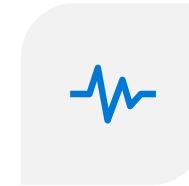
SECURITY SCAN ON
ARTIFACTS



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

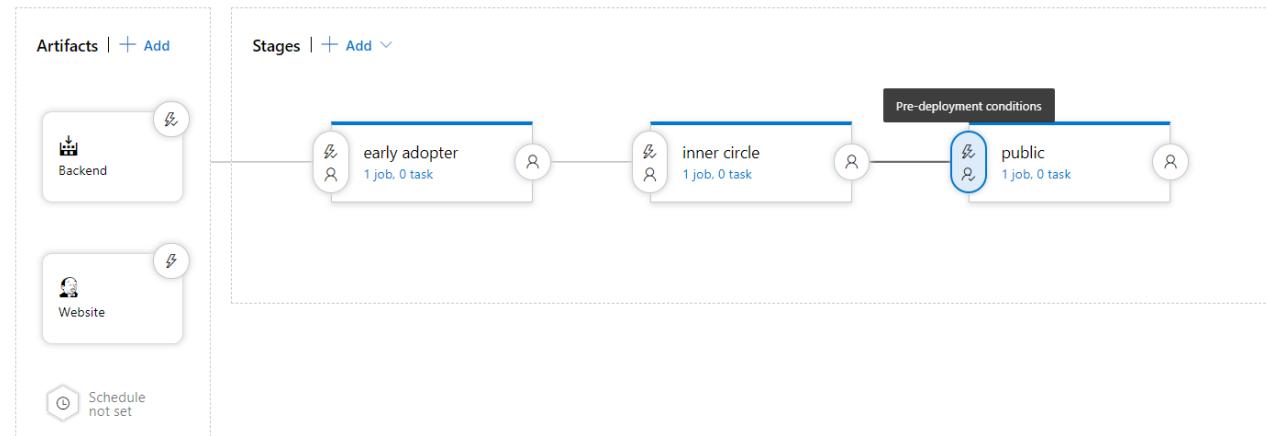
Symptoms of broken process (every second day, only after rerun, never ending up in last stage)

Dashboard widgets

Release Branch Runs - Default

Environments

	Backend	Website	early adopter	inner circle	public	
Sps.SelfTest	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%
Sps.SelfHost	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%
Tfs.SelfHost Set 1	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%
Tfs.SelfHost Set 2	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✗ 98.69%	✓ 100%
Tfs.SelfTest	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%
Tfs.Deploy		✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%
TfsOnPrem.SelfHost	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%
TfsOnPrem.SelfTest	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%	✓ 100%



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

The screenshot shows a work item details page for a feature named "344 Shopping Cart should be personalized". The work item is currently "Unassigned" and has 0 comments. The "Status" is listed as "New" under "State" and "Test demo" under "Area". The "Reason" is "New feature" and the "Iteration" is "Test demo". The "Description" field contains the text "The shopping cart should know the user". The "Acceptance Criteria" field has a placeholder "Click to add Acceptance Criteria". The "Release Notes" field is highlighted with a red border and contains the text "Here can be the commercial release notes.". The "Discussion" field has a placeholder "Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person." On the right side, there are sections for "Development" (status "Development", note "Development hasn't started on this item.", "Add link" button), "Related Work" (note "There are no links in this group."), and "Details" (Priority: 2, Effort, Business Value, Time Criticality, Value area: Business).

FEATURE 344
344 Shopping Cart should be personalized

Unassigned 0 comments Add tag Save Follow ...

Updated by rvanosnabrugge just now

Description Status

The shopping cart should know the user Start Date

Acceptance Criteria Target Date

Click to add Acceptance Criteria Details

Release Notes Priority

Here can be the commercial release notes. 2

Discussion Effort

Add a comment. Use # to link a work item, ! to link a pull request, or @ to mention a person.

Business Value

Time Criticality

Value area Business

Development

+ Add link Development hasn't started on this item.

Related Work

+ Add link There are no links in this group.

Lesson 04: Choosing a Deployment Pattern

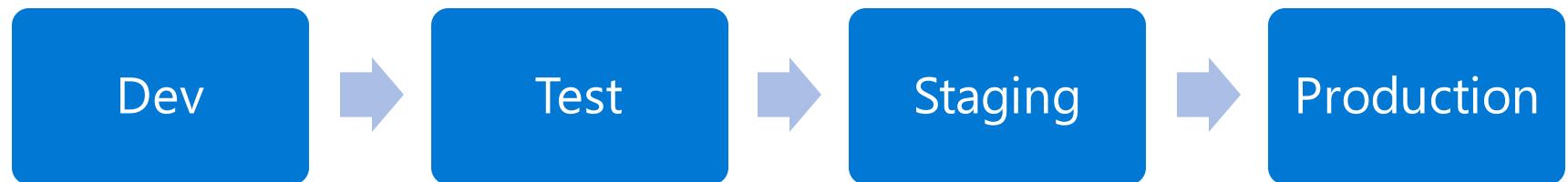


Deployment patterns

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



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

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 executable 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

The screenshot shows the 'Variables' tab in the Azure DevOps interface. The page includes a navigation bar with 'Variables', 'Retention', 'Options', and 'History' tabs. A search bar at the top allows filtering by keywords. On the right, there are 'List' and 'Grid' view options, and a 'Scope' dropdown set to 'Release'. A red box highlights the 'Scope' dropdown and the 'Release' option. Below the table, a lock icon indicates secured variables.

Name	Value
Prefix	Demo
ServerName	DevServer
ServerName	TestServer
Password	*****

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

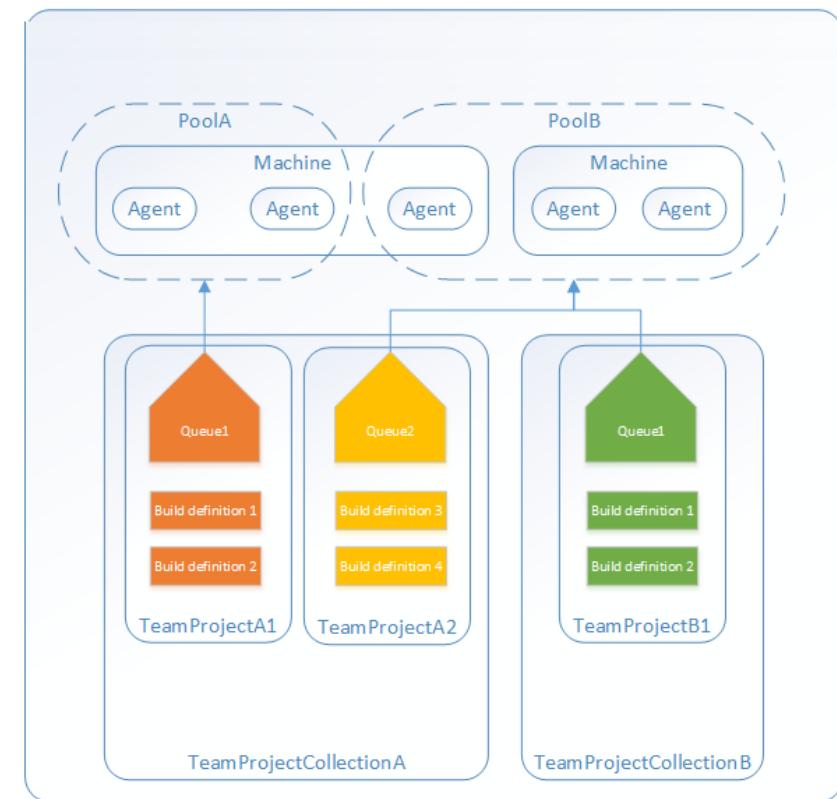
The screenshot shows a user interface for managing build tasks. At the top, there are buttons for 'Add tasks' and 'Refresh', and a search bar. Below this, a navigation bar includes links for 'All', 'Build', 'Utility' (which is underlined, indicating it's the active category), 'Test', 'Package', 'Deploy', 'Tool', and 'Marketplace'. The main content area displays a list of tasks:

- Download Secure File**: Described as "Download a secure file to a temporary location on the build or release agent".
- Extract Files**: Described as "Extract a variety of archive and compression files such as .7z, .rar, .tar.gz, and .zip".
- FTP Upload**: Described as "FTP Upload".
- GitHub Release**: Described as "Create, edit, or delete a GitHub release".
- Install Apple Certificate**: Described as "Install an Apple certificate required to build on a macOS agent".
- Install Apple Provisioning Profile**: Described as "Install an Apple provisioning profile required to build on a macOS agent".
- Install SSH Key**: Described as "Install an SSH key prior to a build or release". This task is highlighted with a light blue background.

At the bottom right of the content area is a large 'Add' button.

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.

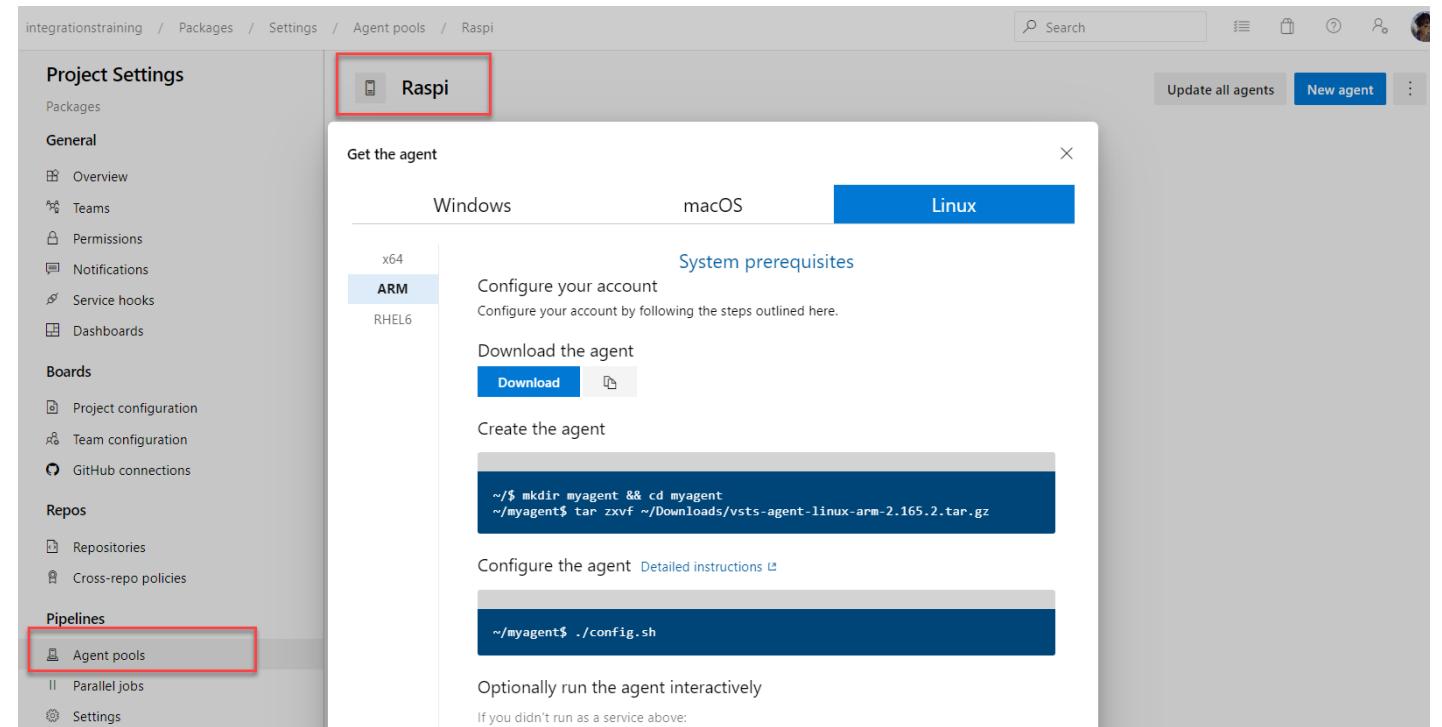
Hosted agents have specific limitations and advantages:

- 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

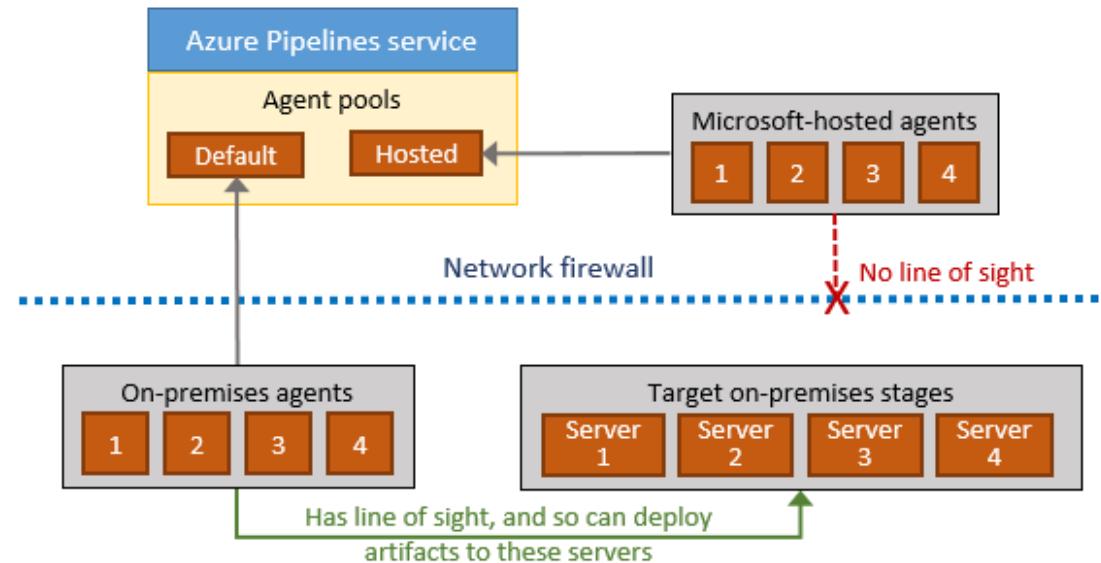
A deployment group is a logical set of deployment target machines that have agents installed on each one.

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

All pipelines > Release Jobs Picture

Pipeline Tasks Variables Retention Options History

Development Deployment process ...

macOS Job + ⋮
Run on agent

Publish to the App Store TestFlight track
Apple App Store Release

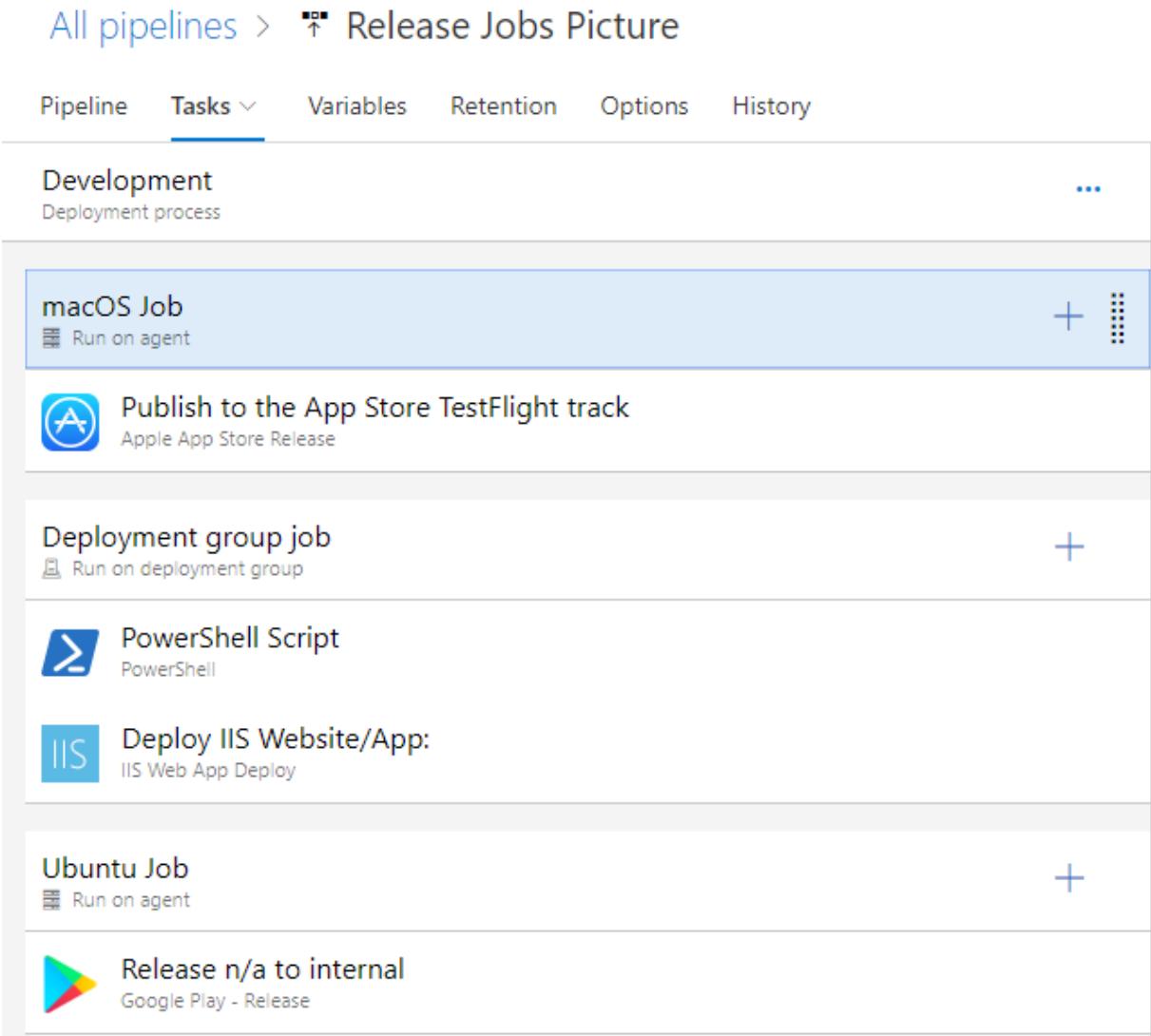
Deployment group job + ⋮
Run on deployment group

PowerShell Script
PowerShell

Deploy IIS Website/App:
IIS Web App Deploy

Ubuntu Job + ⋮
Run on agent

Release n/a to internal
Google Play - Release



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



HANDS-
ON

Lesson 03: Manage Secrets with the Release Pipeline



Manage And Modularize Tasks and Templates

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

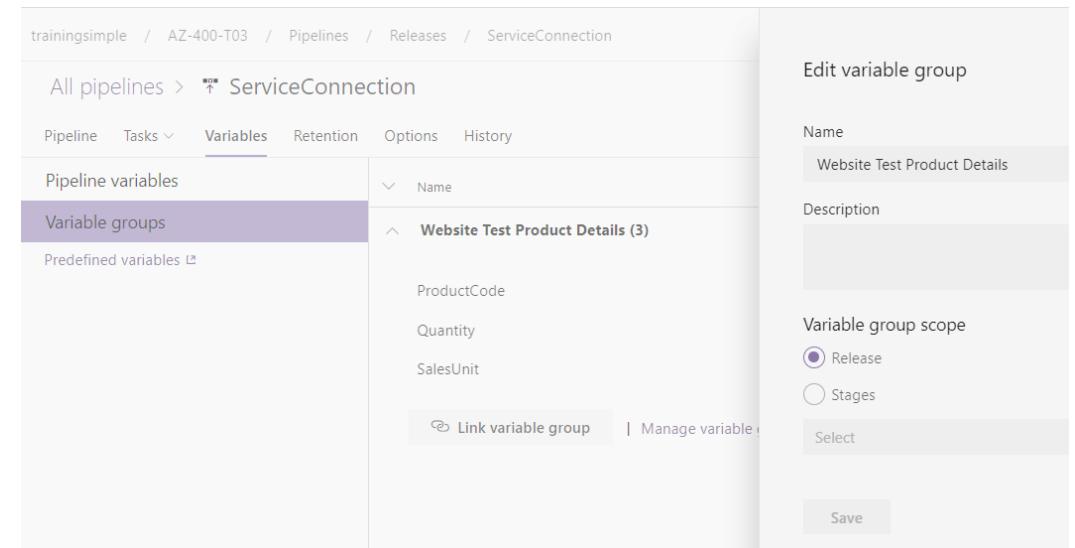
- Task Groups
- Variable Groups
- Custom Build and Release Tasks

Variable Groups

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

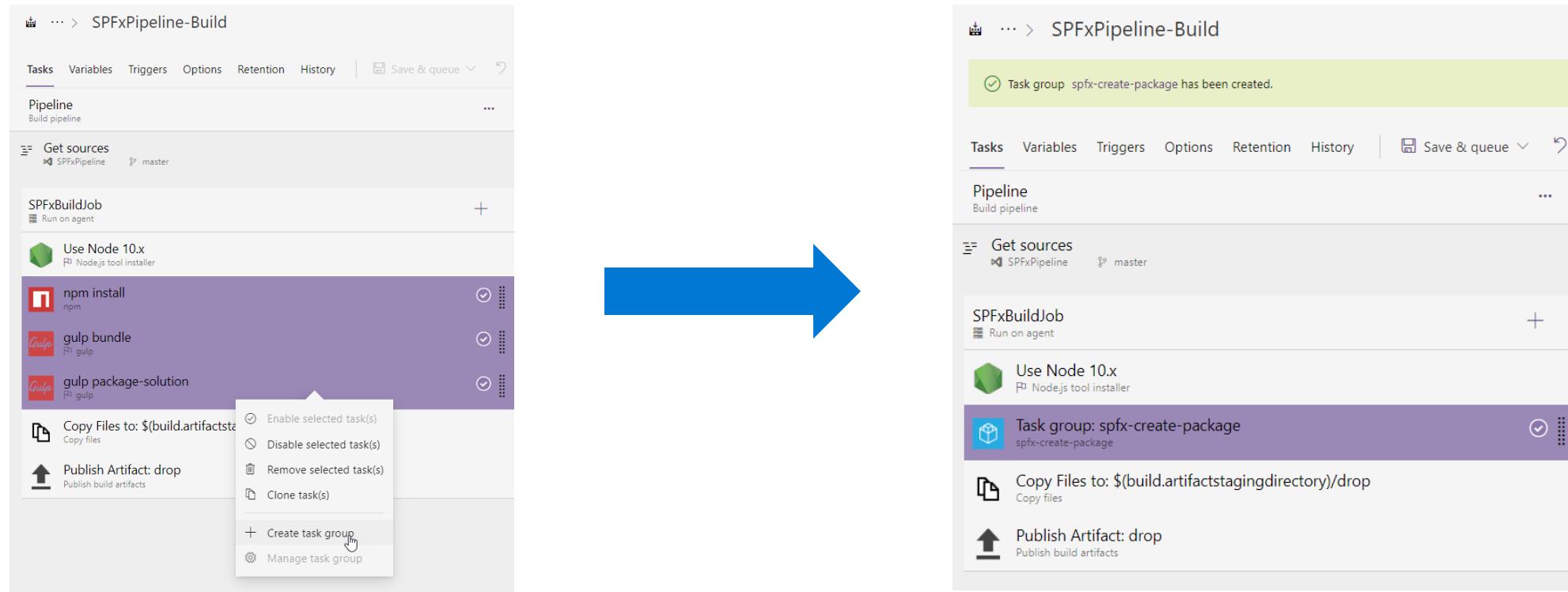
Can be used to:

- 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

```
# File: azure-pipelines.yml

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

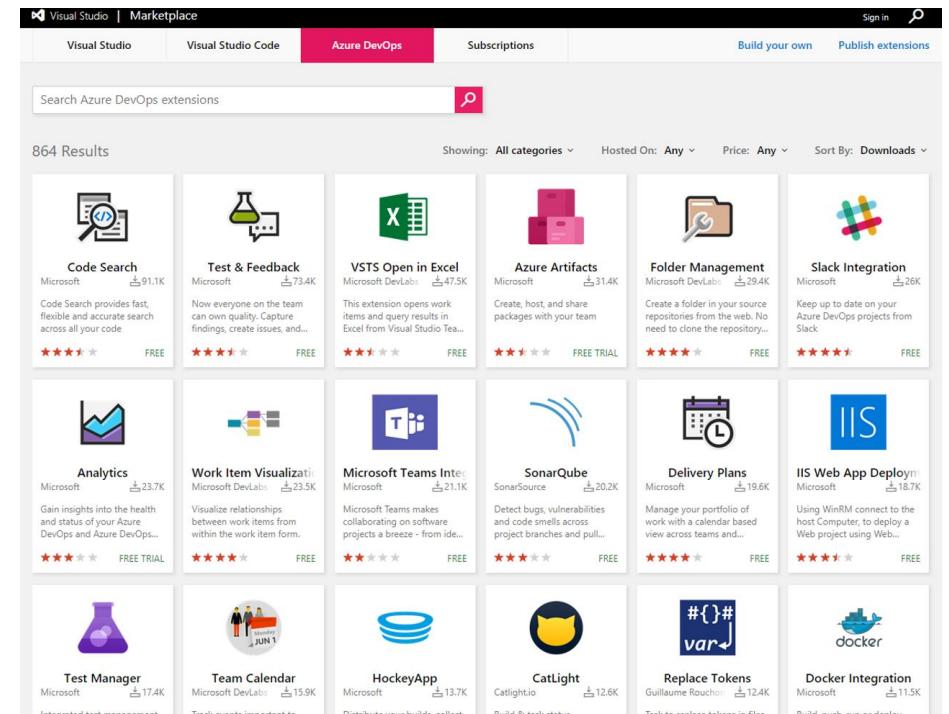
```
# File: jobs/build.yml

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

The screenshot shows the 'Tasks' tab of a pipeline named 'T03-Demos'. The pipeline has three tasks: 'Agent job', 'Backup website zip file', and 'Azure Key Vault: az400t03de...'. The 'Azure Key Vault' task is selected, showing its configuration details. The 'Task version' is set to '1.*'. The 'Display name' is 'Azure Key Vault: az400t03demos'. Under 'Azure subscription', it is set to 'ARM Service Connection' (scoped to resource group 'az-400'). The 'Key vault' is set to 'az400t03demos'. The 'Secrets filter' is set to 'database-login, database-password'.

The screenshot shows the 'Variable group' page for 'KeyVaultVars*'. The 'Variable group name' is 'KeyVaultVars'. There are two toggle switches: 'Allow access to all pipelines' (on) and 'Link secrets from an Azure key vault as variables' (on). Below these are fields for 'Azure subscription' (set to 'ARM Service Connection') and 'Key vault name' (set to 'az400t03demos'). The 'Variables' section shows two entries: 'database-login' and 'database-password', both of which are enabled and have an expiration date of 'Never'. The status is 'Last refreshed: just now'.

Delete	Secret name	Content type	Status	Expiration date
	database-login		Enabled	Never
	database-password		Enabled	Never

Setting up Secrets in the pipeline



Setting up secrets in the pipeline with Azure Key vault



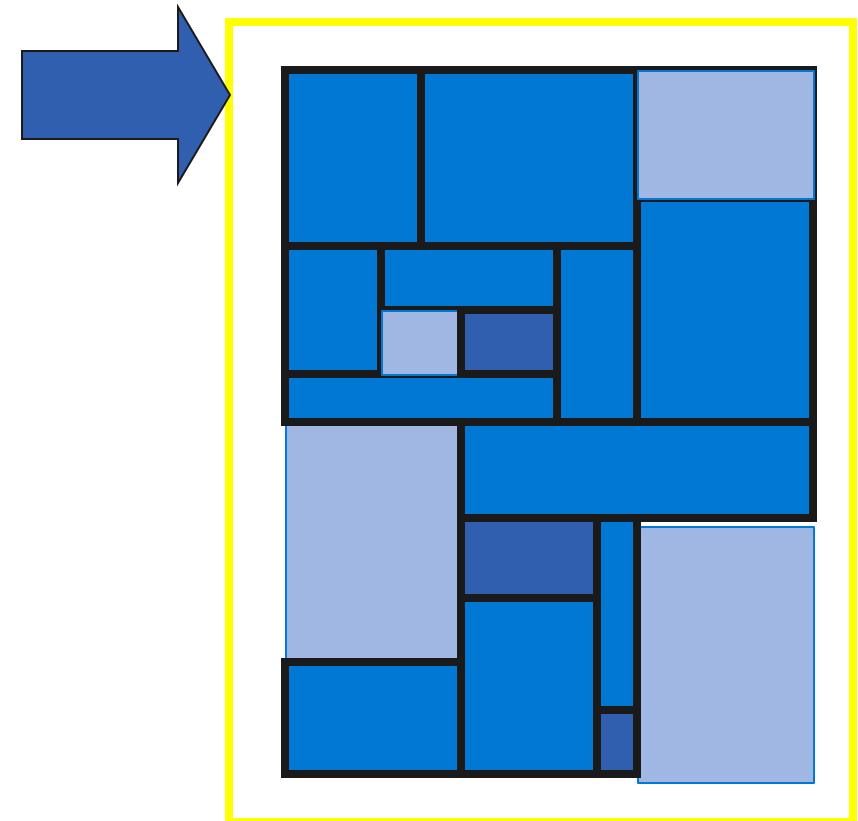
HANDS-
ON

Lesson 05: Configure Automated Integration and Functional Test Automation



Traditional Testing

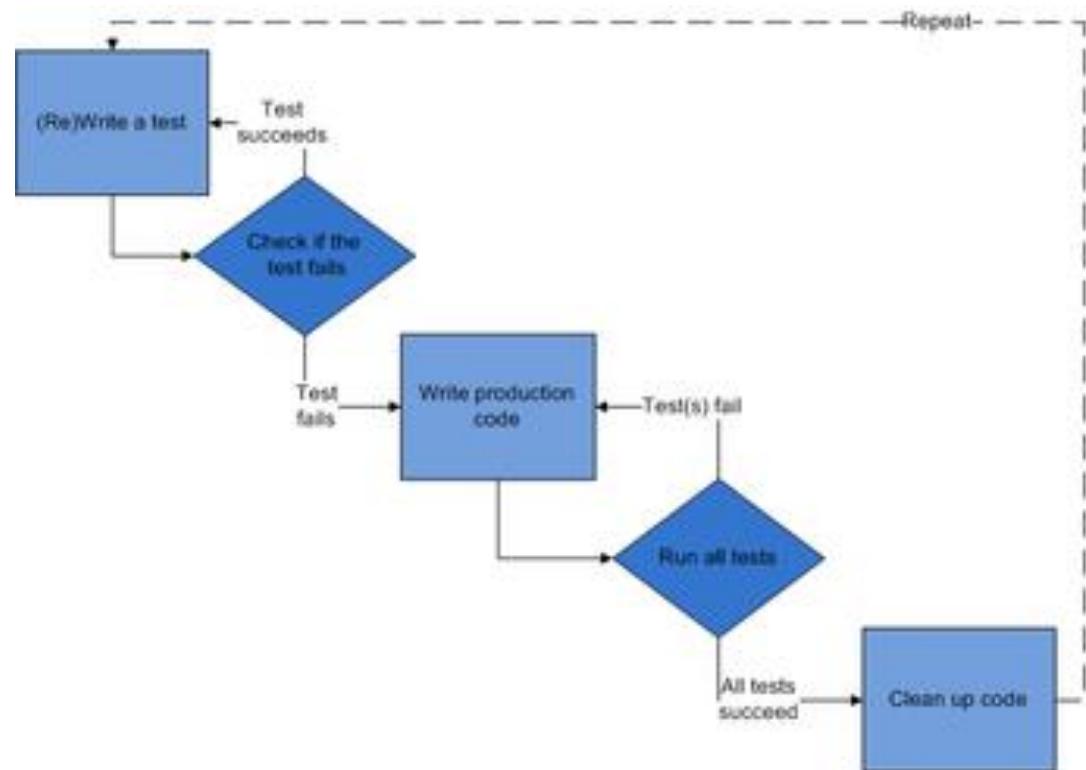
- Test the system as a whole
- Individual components rarely tested
- Errors go undetected
- Isolation of errors difficult to track down



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

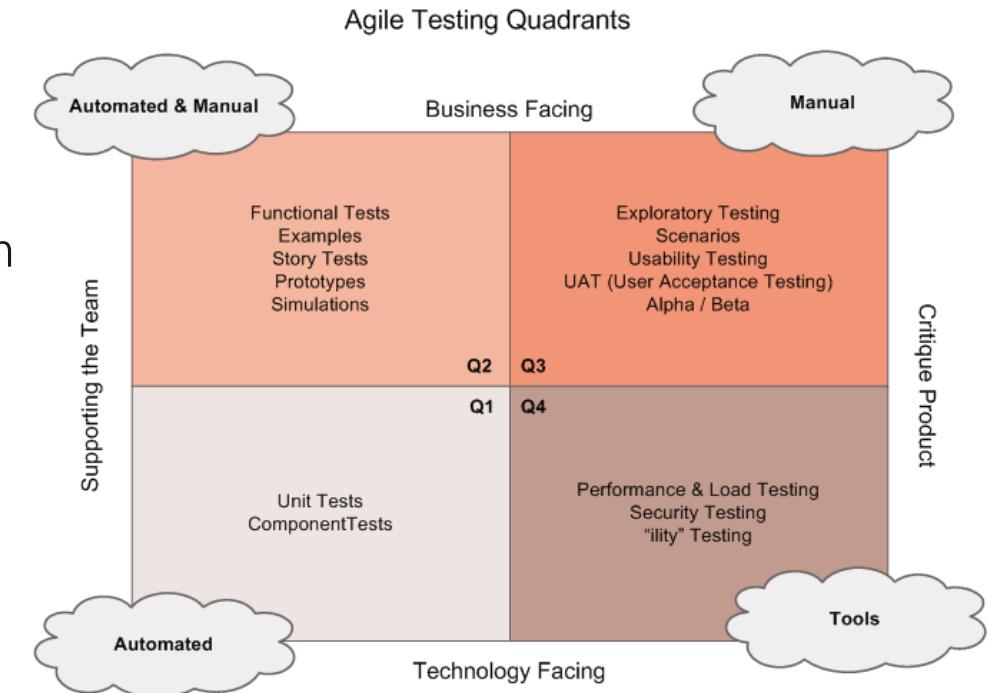


Test Types

- 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



HANDS-
ON

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

The screenshot shows the 'Service hooks' page in the Azure DevOps 'Project Settings' for the 'AZ-400-T03' project. The left sidebar lists various project management sections like General, Teams, Permissions, Notifications, Service hooks (which is selected), Dashboards, Boards, Project configuration, Team configuration, GitHub connections, Repositories, Cross-repo policies, Agent pools, Parallel jobs, Settings, Test management, Release retention, and Service connections*. A modal window titled 'NEW SERVICE HOOKS SUBSCRIPTION' is open, prompting the user to 'Select a service to integrate with.' It includes a 'Discover more integrations' link and a scrollable list of services. The 'Azure App Service' option is highlighted, showing its details: 'Azure App Service is a single service that includes all of the existing capabilities from Azure Websites, Azure Mobile Services, and Azure Biztalk Services, while adding new capabilities. Learn more about how App Service relates to these existing services.' It also lists supported events ('Code pushed') and actions ('Deploy web app'), with a 'Learn more about this service' link.

trainingsimple / AZ-400-T03 / Settings / Service hooks

A Project Settings AZ-400-T03 General Overview Teams Permissions Notifications Service hooks Dashboards Boards Project configuration Team configuration GitHub connections Repositories Cross-repo policies Pipelines Agent pools Parallel jobs Settings Test management Release retention Service connections*

Service Hooks

Integrate with your favorite services by notifying them when events happen in your project.

+ Create subscription

NEW SERVICE HOOKS SUBSCRIPTION

Service

Select a service to integrate with. [Discover more integrations](#)

App Center
AppVeyor
Azuqua
Azure App Service
Azure Service Bus
Azure Storage
Bamboo
Campfire
Datadog
Flowdock
Grafana
HipChat
HockeyApp
Jenkins

Azure App Service

Azure App Service is a single service that includes all of the existing capabilities from Azure Websites, Azure Mobile Services, and Azure Biztalk Services, while adding new capabilities. [Learn more](#) about how App Service relates to these existing services.

Supported events:
Code pushed

Supported actions:
Deploy web app

[Learn more about this service](#)

Previous Next Test Finish Cancel

Release Dashboards

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

Screenshot of the Azure DevOps Release Dashboard for the team "AZ-400-T03".

The dashboard includes the following sections:

- Welcome:** Get started using Azure DevOps to make the most of your team dashboard.
- Work assigned to Alexander Pajer:** Last time you checked, there were no results.
- Critical Bugs:** 1 Work items
- Feedback:** 0 Work items
- Sprint 2:** 25. März - 15. April
- Work in Progress:** 1 Work items
- Unfinished Work:** 4 Work items
- AZ-400-T03 Team:** Changing the world? Make your dream team. Invite a friend.
- User Stories (18):** A table showing user stories assigned to the team. The table includes columns for ID, Work item type, Title, Assigned to, State, and a View query link.
- User Stories by Assigned To:** A bar chart showing the count of user stories assigned to different team members. The x-axis shows the number of stories from 2 to 8, and the y-axis shows the count from 0 to 20. Legend: (blank) (blue), craig109@ou... (orange), ckelly109@ou... (grey).
- User Stories by State:** A donut chart showing the distribution of user stories by state. Total: 18. Legend: New (blue), Committed (orange), Approved (grey), Done (yellow).
- Work:** Backlog, Board, Task board, Queries.
- New Work Item:** Enter title, Bug dropdown, Create button.

Using Azure Monitor as Release Gate



HANDS-
ON

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?

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

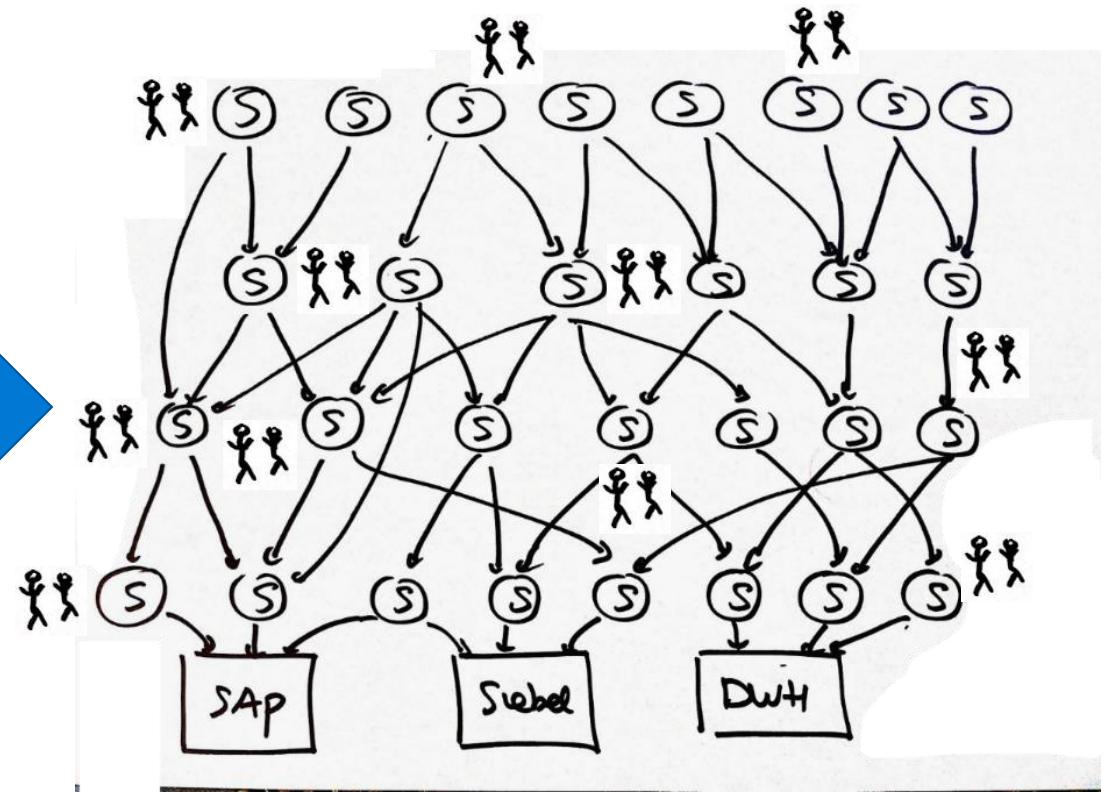
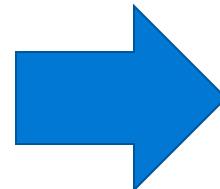
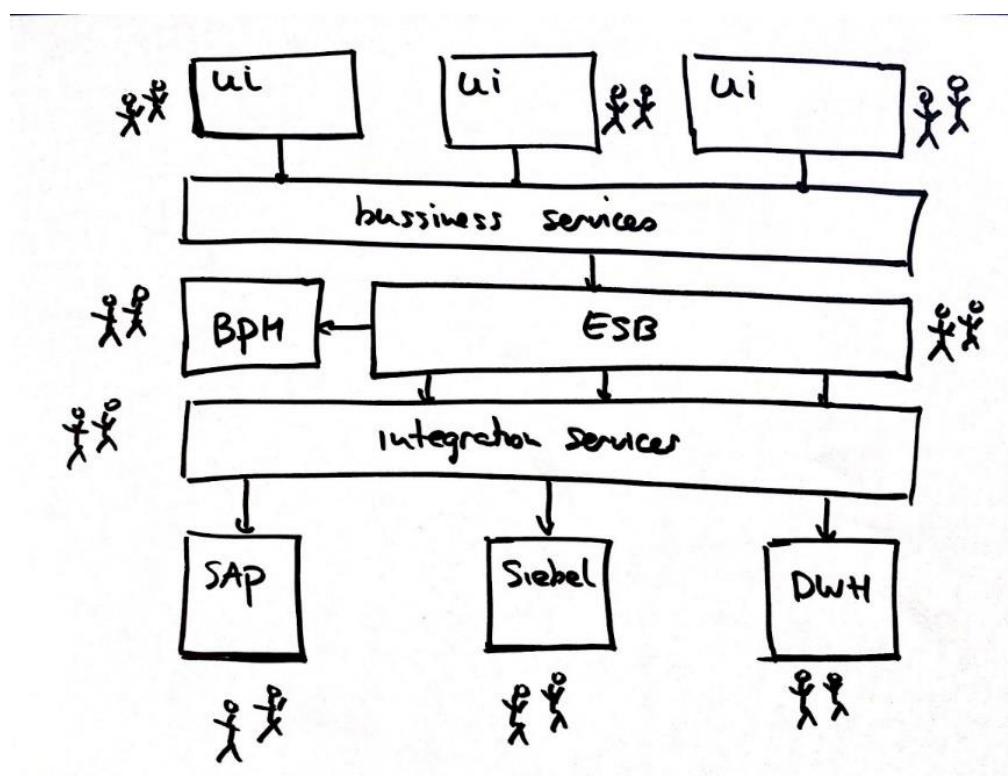
- Break up in smaller pieces
- Microservices

Definitions of Microservices

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

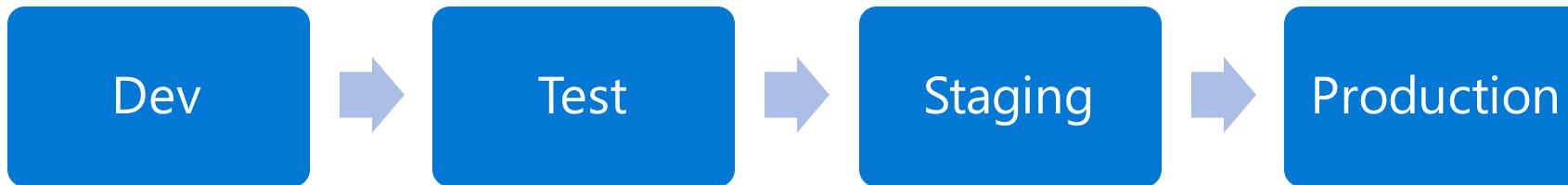
- Services in a microservice architecture (MSA) are often processes that communicate over a network to fulfil a goal using technology-agnostic protocols such as HTTP.^{[2][3][4]}
- Services in a microservice architecture are independently deployable.^{[5][6]}
- Services are organized around business 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

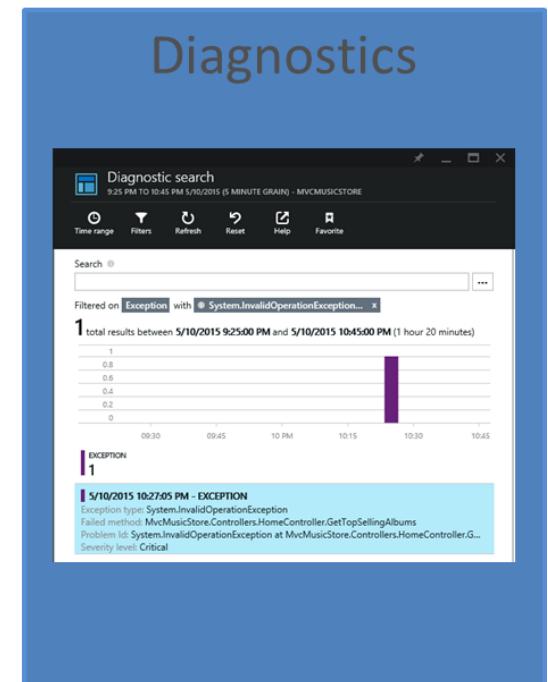
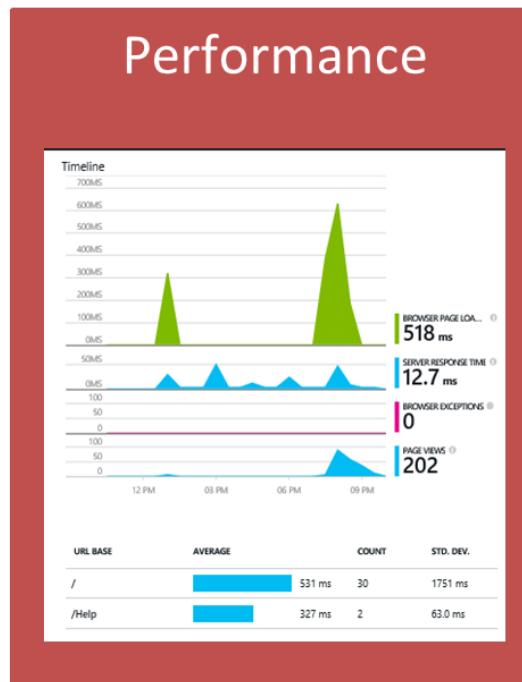


Modern Practices:

- 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 business needs) - Im Einklang sein

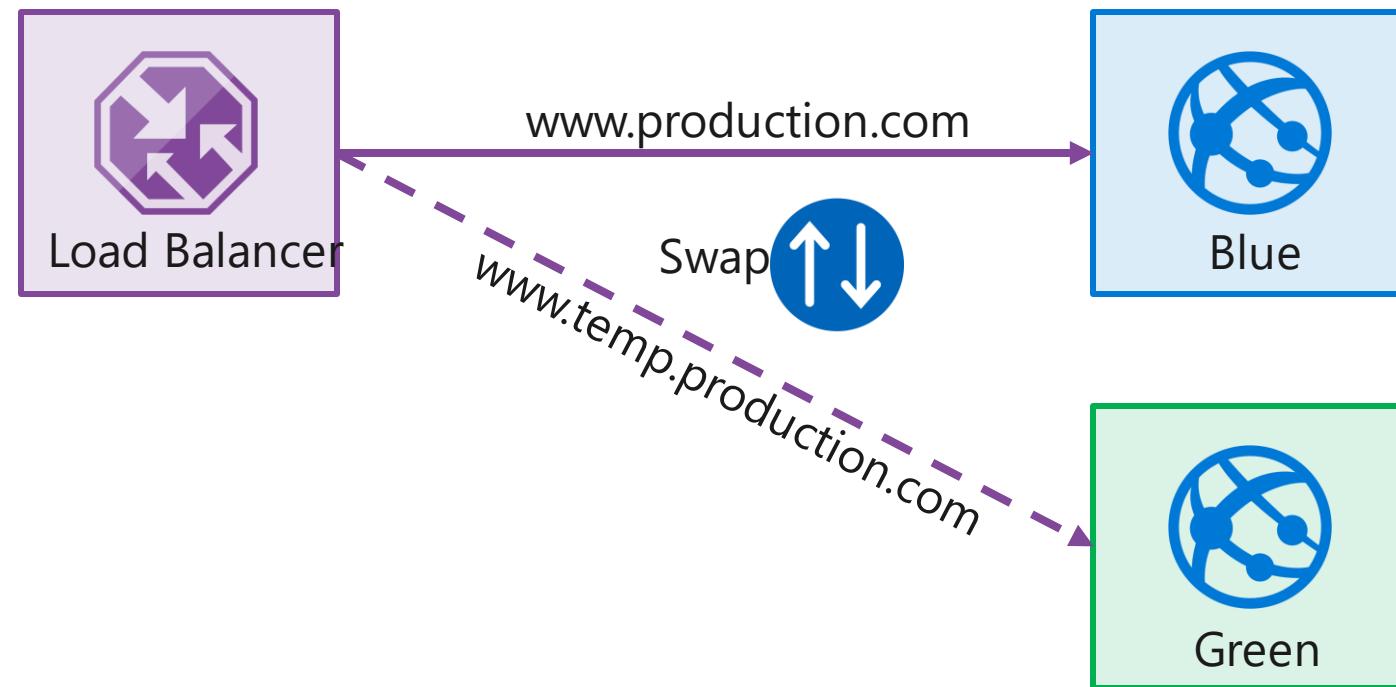


Lesson 02: Implement Blue Green Deployment



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.

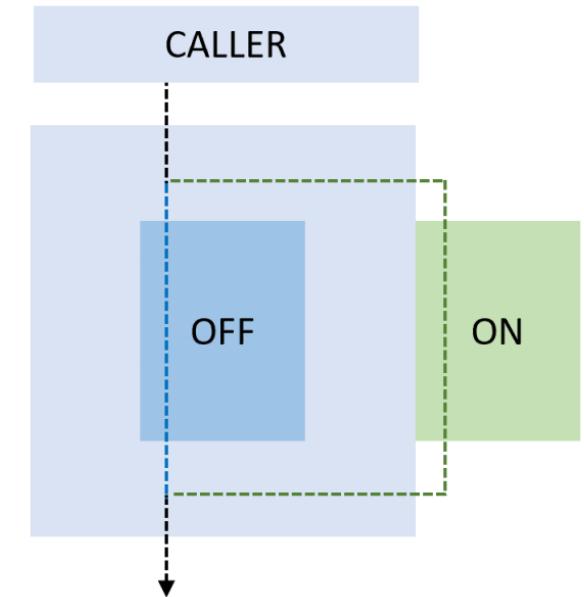
Why do you need it?

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

The screenshot shows the Azure App Configuration interface for managing feature toggles. On the left, there's a sidebar titled "App Configuration" with options like "Add", "Manage view", and a search bar. Below it is a list of configurations, with one named "AZ-400-Demo" highlighted. A vertical dashed line separates the sidebar from the main content area. The main area has a title "AZ-400-Demo - Feature manager" and a sub-header "App Configuration". It includes a search bar, "Add" and "Refresh" buttons, and a message "Total 1 feature flags loaded". A table lists the feature flag details:

Key	Label	State	Description
Rating	Rating Feature	<input type="button" value="Off"/> <input type="button" value="On"/>	

A red box highlights the "Feature manager" option in the sidebar's "Operations" menu.

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?

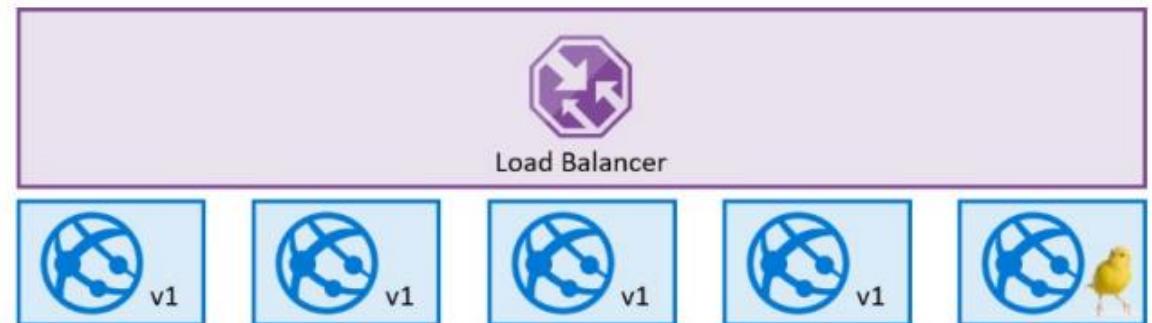
Traffic manager provides the ability to route traffic between Azure app services

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.

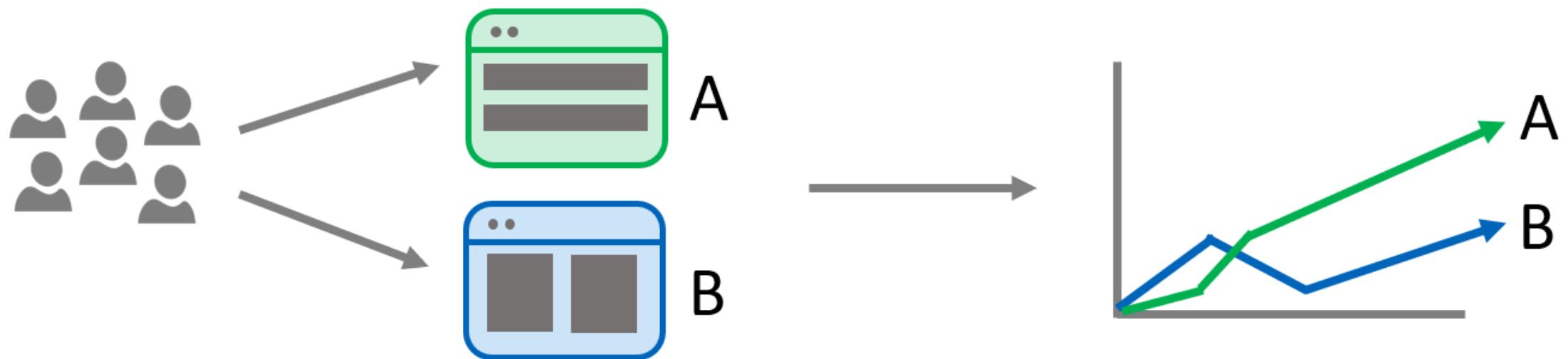
Usually, these users aren't aware they are being used as guinea pigs for the new feature and often you don't even highlight the new feature to them, hence the term "Dark" launching.

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

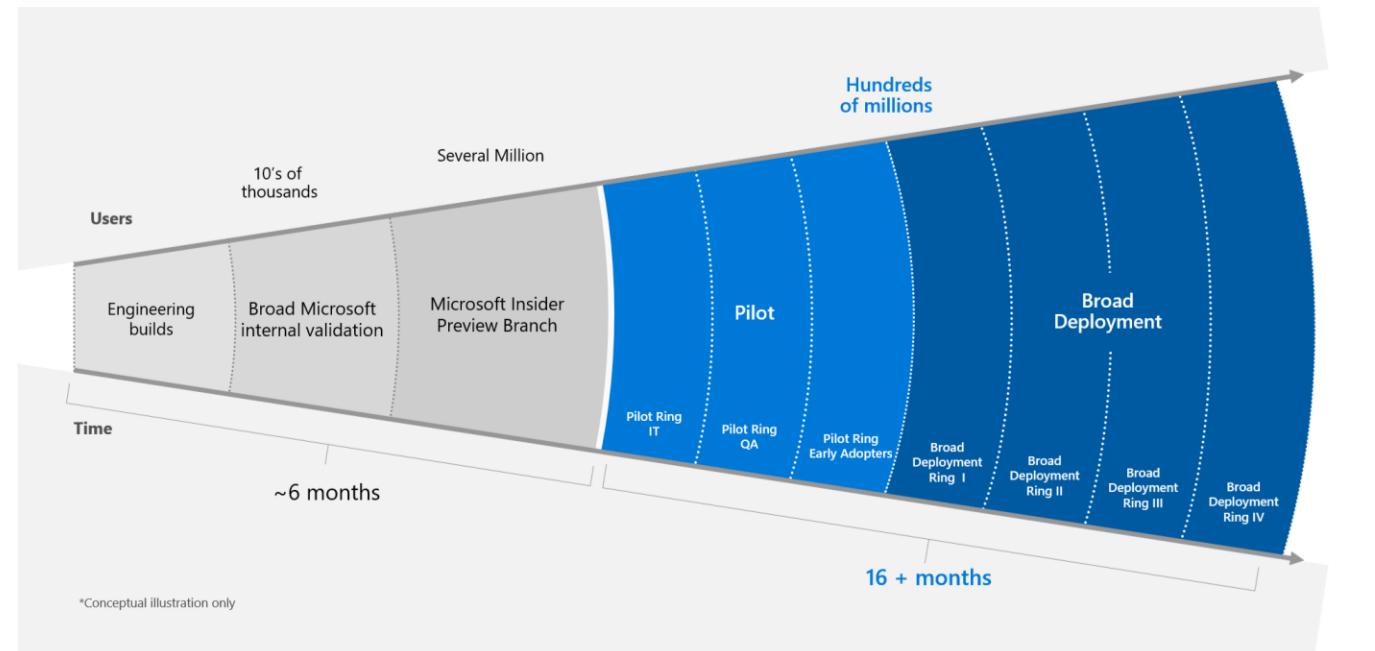
- Experiment on features and usage
- Improve conversion rate
- Continuous experiments
- Measure impact of change

Lesson 07: Progressive Exposure Deployment

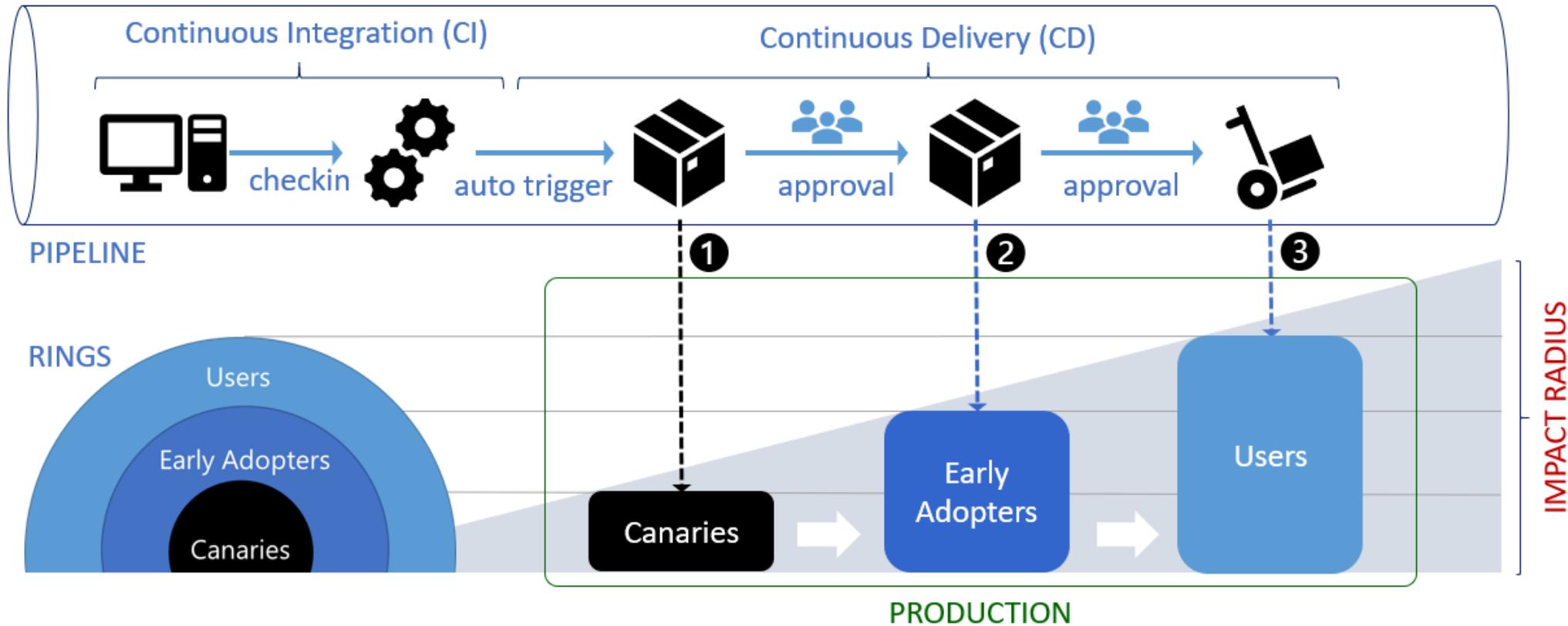


What is Progressive Exposure Deployment

- 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

