

# AZ-400.00

## Learning Path 04: Design and implement a release strategy



# Agenda



- Module 01: Introduction to continuous delivery.
- Module 02: Create a release pipeline.
- Module 03: Explore release strategy recommendations.
- Module 04: Provision and test environments.
- Module 05: Manage and modularize tasks and templates.
- Module 06: Multi-stage YAML.
- Module 07: Automate inspection of health.
- Labs & Learning Path review and takeaways.



# Learning Path overview

# Learning objectives

After completing this Learning Path, students will be able to:

- 1 Explain the terminology used in Azure DevOps and other release management tooling
- 2 Describe what a build and release task is, what it can do, and some available deployment tasks
- 3 Explain why you sometimes need multiple release jobs in one release pipeline
- 4 Differentiate between a release and a deployment
- 5 Define the components of a release pipeline
- 6 Use release variables and stage variables in your release pipeline
- 7 Deploy to environment securely using a service connection
- 8 List the different ways to inspect the health of your pipeline and release by using alerts, service hooks, and reports
- 9 Explain things to consider when designing your release strategy

# Module 01: Introduction to continuous delivery



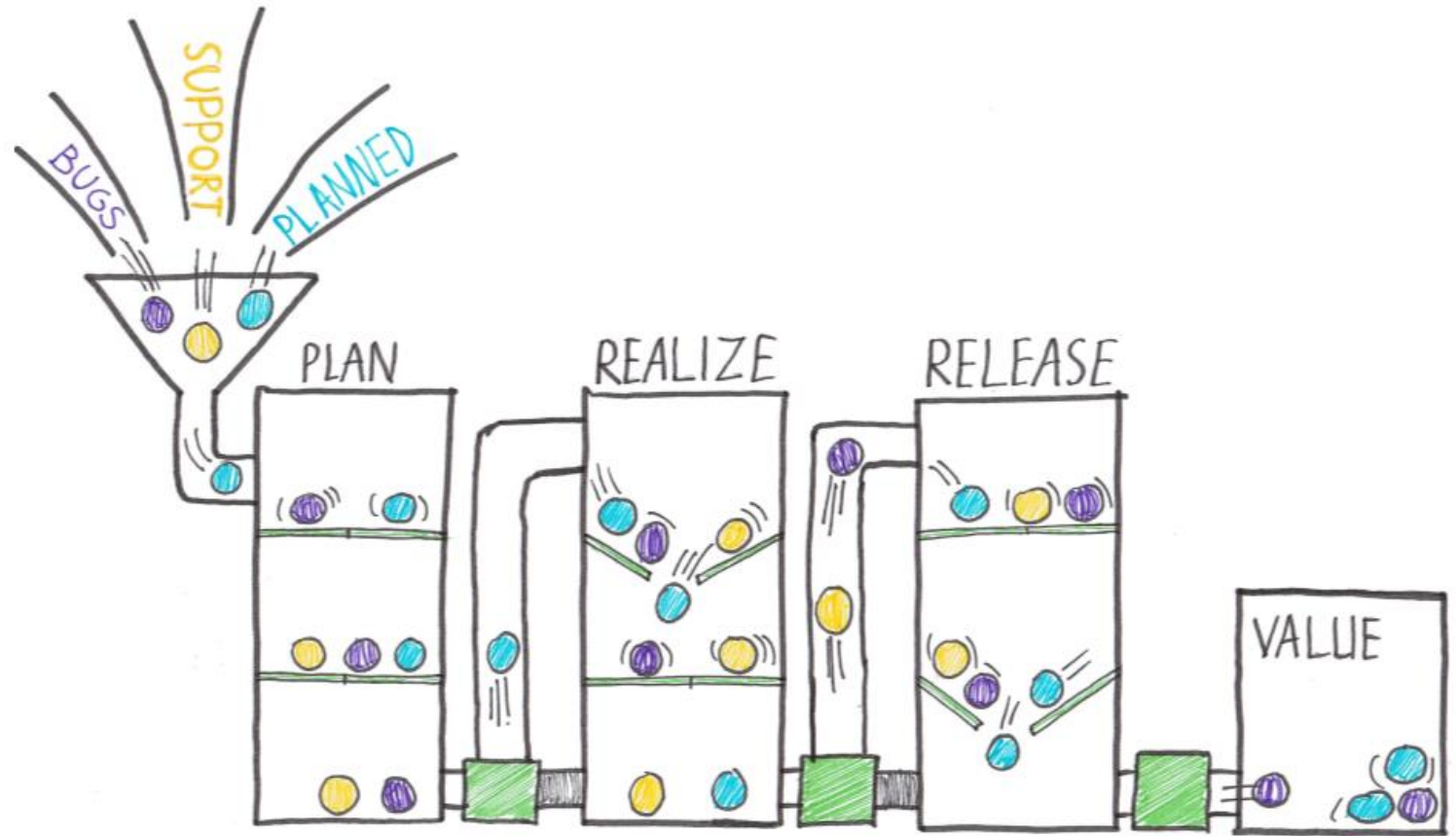
# Learning Objectives

After completing this Learning Path, students will be able to:

- 1 Explain the terminology used in Azure DevOps and other release management tooling
- 2 Describe what a build and release task is, what it can do, and some available deployment tasks
- 3 Explain why you sometimes need multiple release jobs in one release pipeline
- 4 Differentiate between a release and a deployment
- 5 Define the components of a release pipeline
- 6 Use release variables and stage variables in your release pipeline
- 7 Deploy to environment securely using a service connection
- 8 List the different ways to inspect the health of your pipeline and release by using alerts, service hooks, and reports
- 9 Explain things to consider when designing your release strategy

# Explore traditional IT development cycle

- A fair number 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



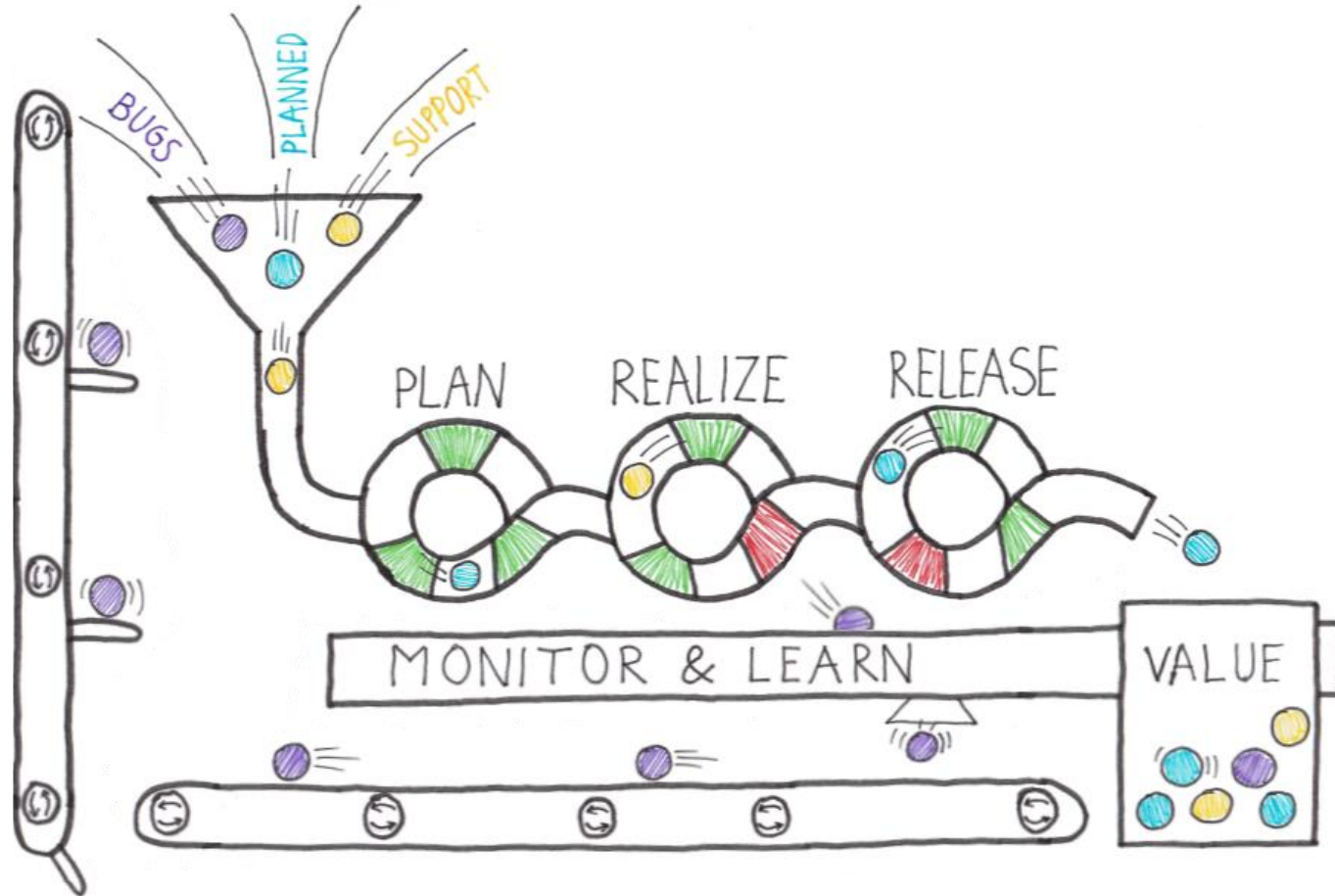
# What is continuous delivery?

## The eight principles of continuous delivery:

- The process for releasing/deploying software MUST be repeatable and reliable
- Keep everything in source control
- Everybody has responsibility for the release process
- Automate everything!
- Done means “released”
- Improve continuously
- If something is difficult or painful, do it more often
- Build quality in!



# Move to continuous delivery



# Understand releases and deployments



Release and deployment are often coupled

---



Release is not the same as a deployment

---



## **Separate functional release from technical release:**

Functional release is exposing features to customers

Technical release is deploying functionality

# Understand release process versus release



The release process involves all the steps that you go through when you move your artifact that comes from one of the artifact sources.

---



A release is a package or container that holds a versioned set of artifacts specified in a release pipeline in your CI/CD process. It holds all the information required to carry out all the tasks and actions in the release pipeline, such as the stages (or environments), the tasks for each one, values of task parameters and variables, the release policies such as triggers, approvers, and release queuing options.

---



There can be multiple releases from one release pipeline (or release process).

# Discussion – The need for continuous delivery in your organization

- Does your organization need continuous delivery?
- Do you use agile/scrum?
  - Is everybody involved or only the Dev departments?
- Can you deploy your application multiple times per day? Why or why not?
- What is the main bottleneck for continuous delivery in your organization?
  - The Organization
  - Application Architecture
  - Skills
  - Tooling
  - Tests
  - Other things?

# Module 02: Create a release pipeline



# Describe Azure DevOps release pipeline capabilities



Support pipelines as code (via YAML)

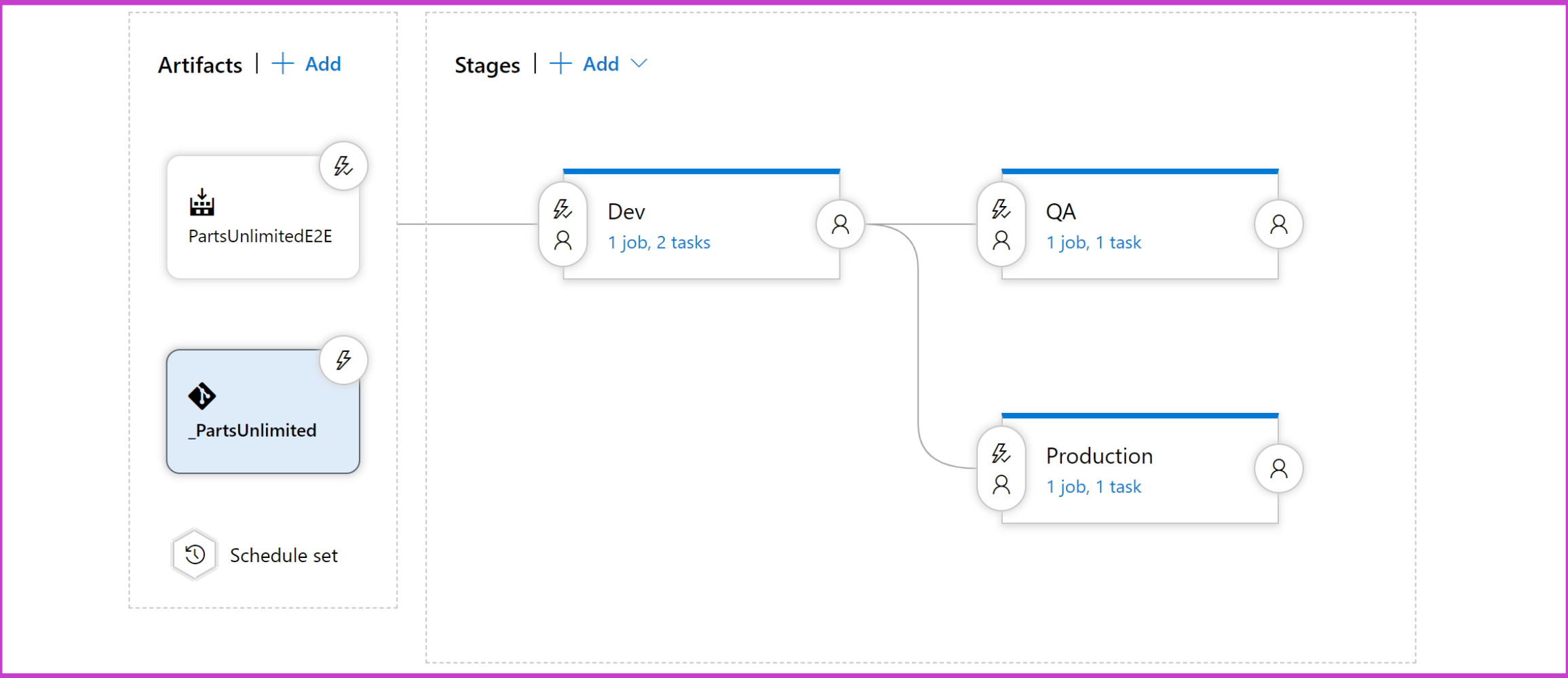
---



Most capabilities from classic release pipelines are available

---

# Explore release pipelines



# Explore artifact sources



Build artifacts

---



Package repositories

---



Container repositories

---



Source control



# Choose the appropriate artifact source



Traceability and auditability

---



Immutability

---



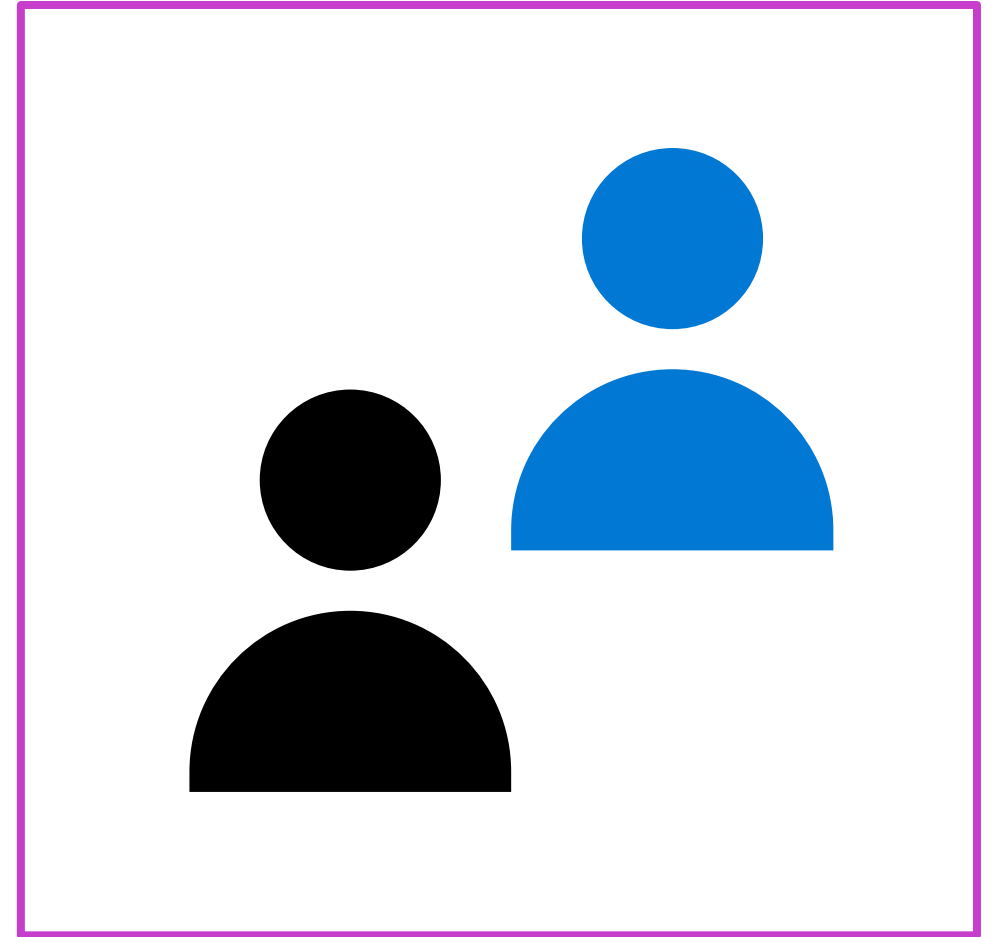
Versioning

# Demonstration: Selecting an artifact source

**DEMO**

# Examine considerations for deployment to stages

- 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?



# Deployment stages

## 1 What is a stage?

Different terms used in various release tools (stages/environment)

A logical boundary in a pipeline that contains one or more jobs

## 2 What is your target environment?

## 3 Long-lived or short-lived environments

## 4 Purpose of an environment

## 5 Feature release and bugfix

# Setting up deployment stages

**DEMO**

# Explore 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


Add tasks

 Refresh


 Search


All Build Utility Test Package Deploy Tool Marketplace


 **Download Secure File**  
Download a secure file to a temporary location on the build or release agent


 **Extract Files**  
Extract a variety of archive and compression files such as .7z, .rar, .tar.gz, and .zip.

 **FTP Upload**  
FTP Upload

 **GitHub Release**  
Create, edit, or delete a GitHub release.

 **Install Apple Certificate**  
Install an Apple certificate required to build on a macOS agent

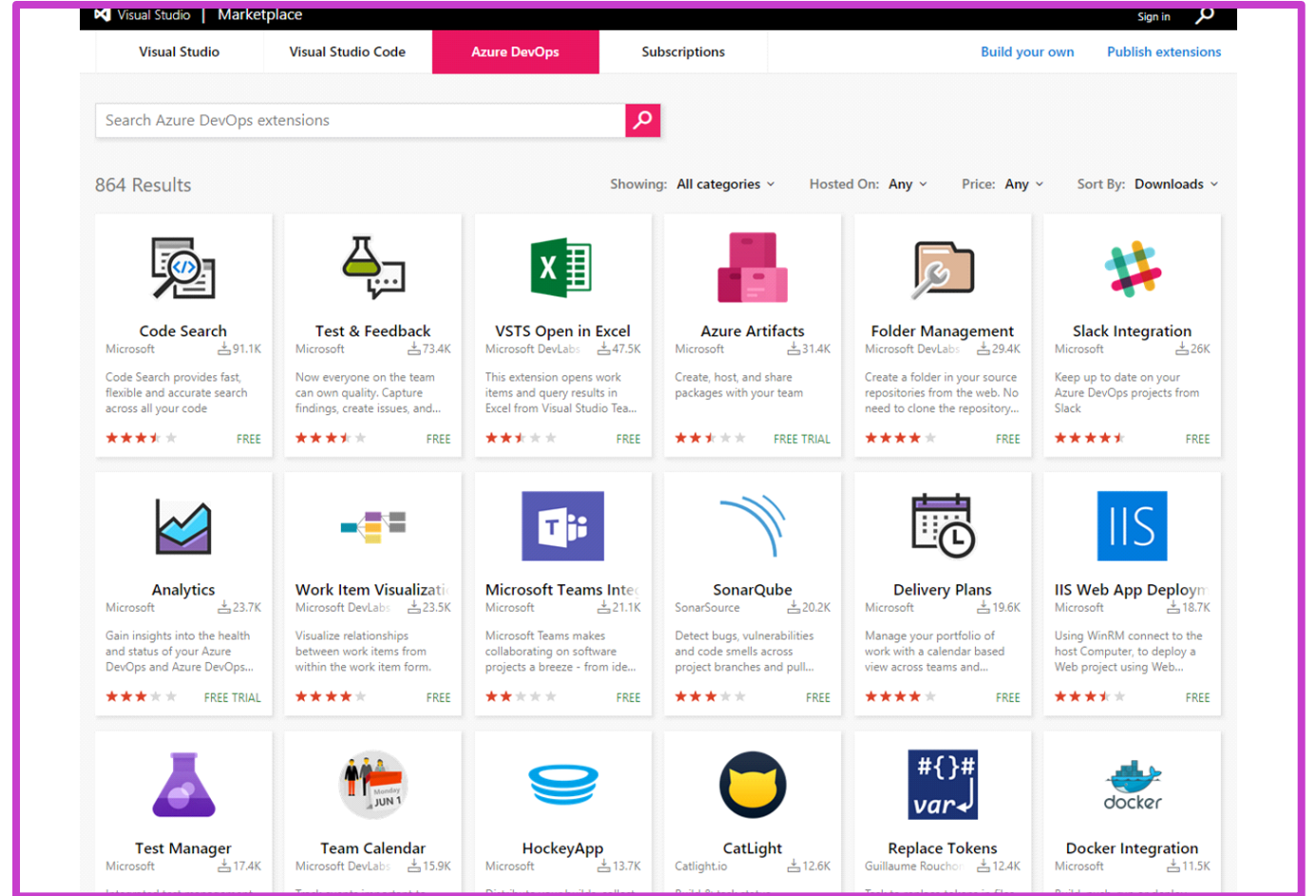
 **Install Apple Provisioning Profile**  
Install an Apple provisioning profile required to build on a macOS agent

 **Install SSH Key**  
Install an SSH key prior to a build or release

Add

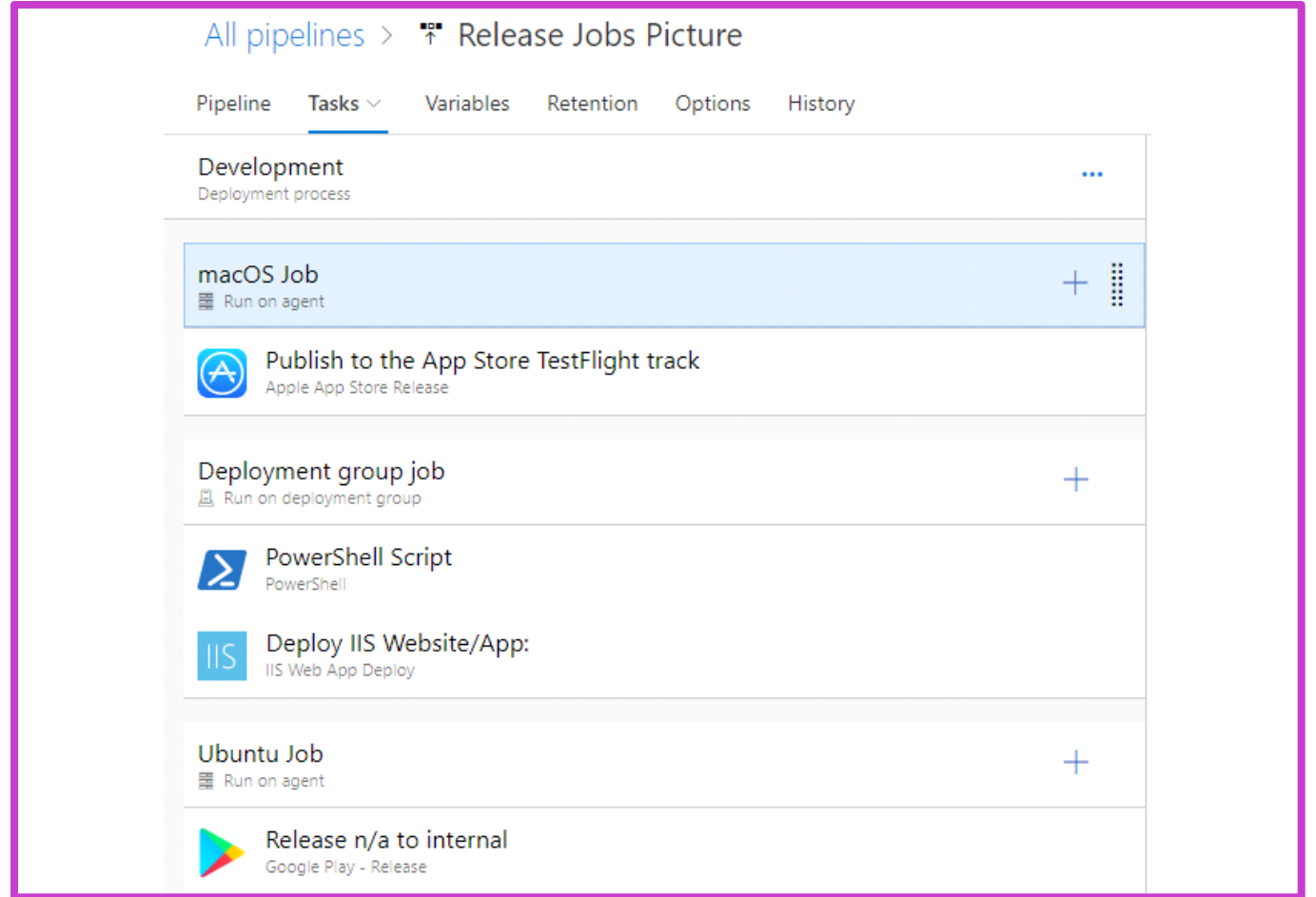
# Explore custom build and release tasks

- 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



# Explore release 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





# Discussion: How to use release jobs

Do you see a purpose for release jobs in your pipeline? How would you set it up?

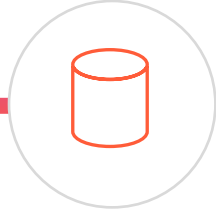
Topics you might want to consider are:

- Do you have artifacts from multiple sources?
- Do you want to run deployments on different servers simultaneously?
- Do you need multiple platforms?
- How long does your release take?
- Can you run your deployment in parallel or does it need to run in sequence?

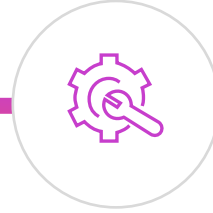
# Module 03: Explore release strategy recommendations



# Understand the delivery cadence and three types of triggers



Continuous  
deployment trigger



Scheduled  
trigger



Manual  
trigger

# Selecting your delivery and deployment cadence

**DEMO**

# Explore release approvals



Release approvals are not to control **how**, but control **if** you want to deliver multiple times a day

---



Manual Approvals help in building trust about the automated release process

---



Release gates give you additional control over the start and completion of the deployment pipeline. They can usually be set up as a pre-deployment and post-deployment condition and can perform validation with other automated systems until specific requirements are verified.

# Demonstration: Setting up manual approvals

**DEMO**

# Explore release gates

Release gates give you additional control over the start and completion of the deployment pipeline. They are often set up as a pre-deployment and post-deployment conditions.

- 1 Incident and issues management
- 2 Notification of users by integration with collaboration systems
- 3 Quality validation
- 4 Security scan on artifacts
- 5 User experience relative to baseline
- 6 Change management
- 7 Infrastructure health

# Use 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?



# Setting up a release gate

**DEMO**

# Module 04: Provision and test environments



# Provision and configure target environments



On-premises servers

---



Cloud servers or Infrastructure as a Service (IaaS). For example, virtual machines or networks

---



Platform as a Service (PaaS) and Functions as a Service (FaaS). For example, Azure SQL Database in both PaaS and Serverless options

---



Clusters

---



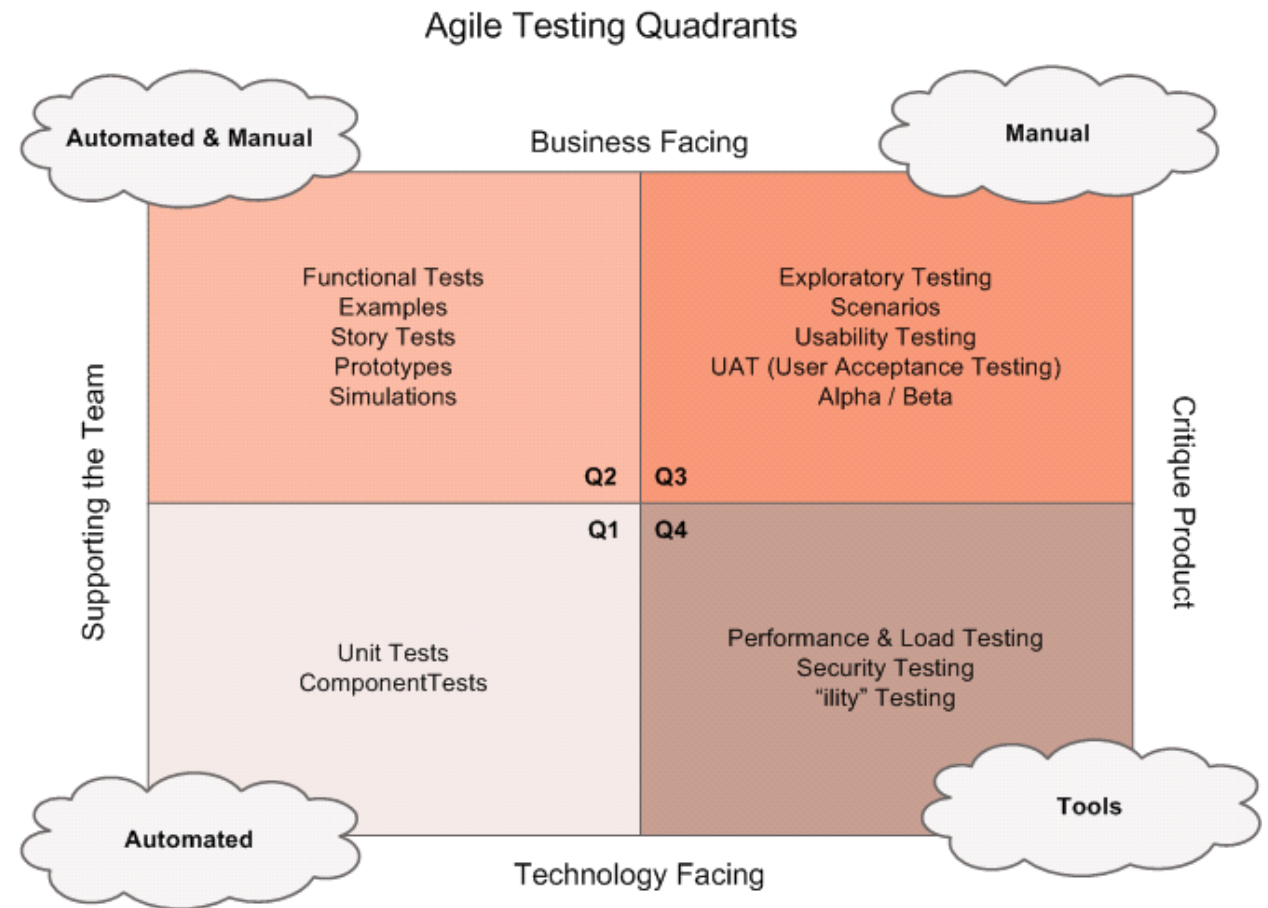
Service connections

# Demonstration: Set up service connections

**DEMO**

# Configure automated integration and functional test automation

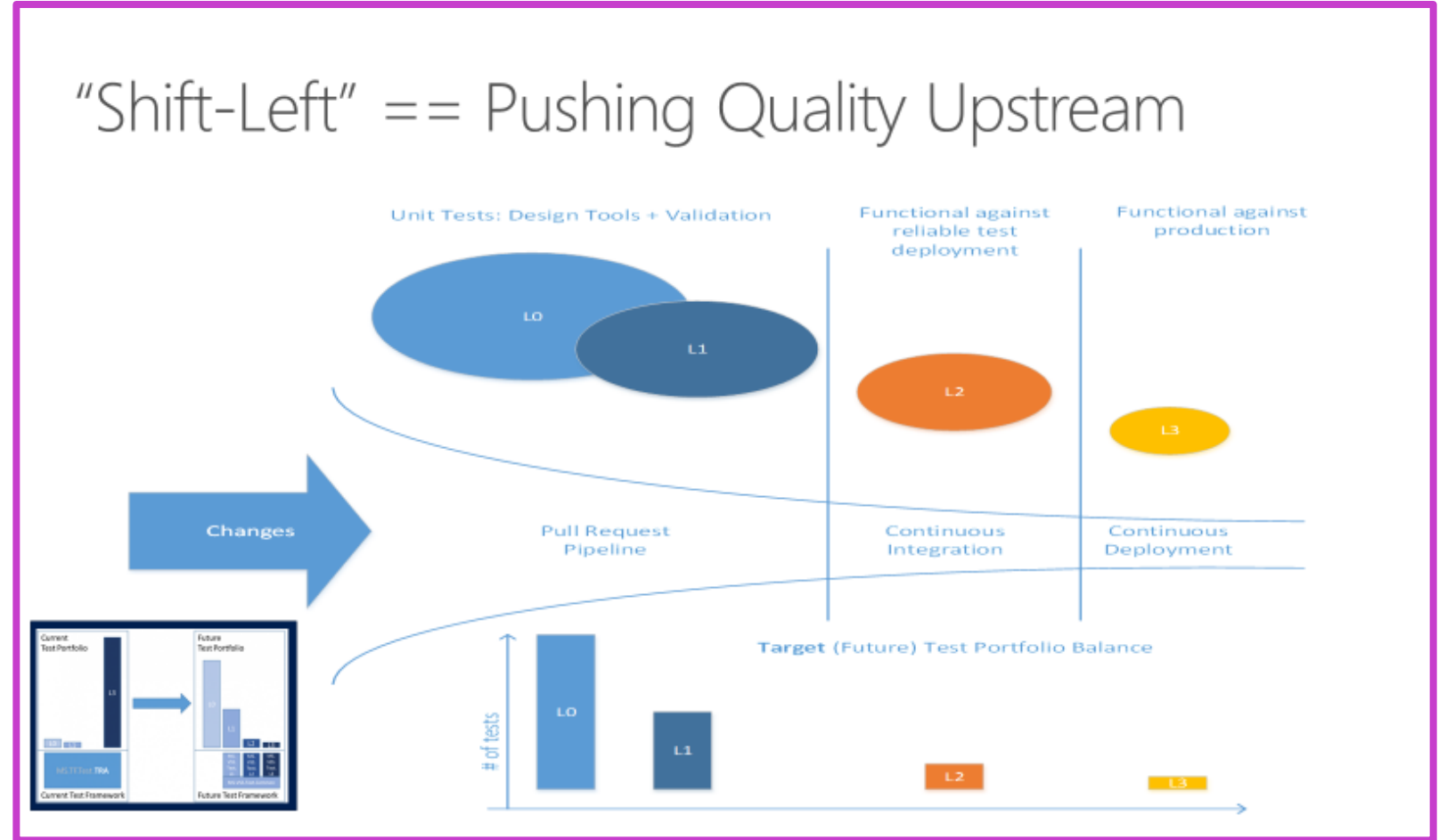
- Do not automate all your manual tests
- Rethink test strategy
- Tests should be written at the lowest level possible
- Write once, run anywhere including production system
- Product is designed for testability
- Test code is product code, only reliable tests survive
- Test ownership follows product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



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

# Understand Shift-left

- The goal for shifting left is to move quality upstream by performing tests early in the pipeline.
- Combine test and process improvements to reduce the time it takes for tests.



# Set up and run availability tests



Create health end points in your application

---



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

---

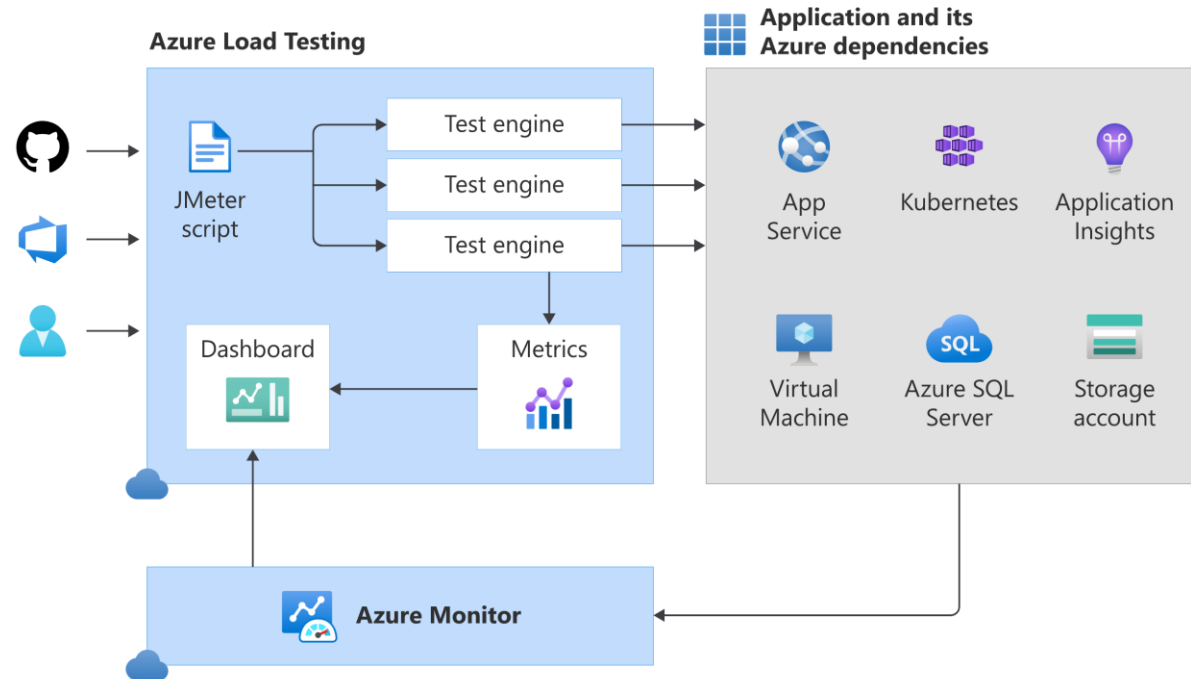


**Two common types of availability tests:**

URL ping test

Multi-step web test

# Explore Azure Load Testing



Azure Load Testing is a fully managed load-testing service that enables you to generate high-scale load.

Integrate Azure Load Testing in your CI/CD pipeline during the development lifecycle.



# Module 05: Manage and modularize tasks and templates



# Examine task groups

The screenshot shows the 'Task groups > Demo' configuration page. On the left, a list of tasks includes 'Azure Deployment' and 'Azure App Service Deploy'. The main area displays the configuration for the 'Demo' task group, version 1.\*. The configuration includes a 'Name' field set to 'Demo', a 'Description' field, a 'Category' dropdown set to 'Deploy', and a 'Parameters' section with a table of parameters.

Name	Default value	Description
HostingPlan	DemoHostingPlan	
ResourceGroupName	mcourse-cd	Provide the name of a res...
ServerName	mcoursecdserv	
slot		Enter or Select an existin...
WebsiteName	mcoursecdserv	

The screenshot shows the task group execution interface. It features a search bar with the text 'demo' and a close button. Below the search bar, there is a section titled 'Add tasks' with a 'Refresh' button. A task group named 'Demo' is listed with a blue icon.

Encapsulate a sequence of tasks in one reusable task

Parameterization of task groups to make reuse easier

Standardize and centrally manage deployment steps for all your applications

Note: Task groups are not currently supported in YAML. Use templates instead.

# Demonstration: Creating and managing task groups

**DEMO**

# Explore variables in release pipelines

Variables

Retention

Options

History

Filter by keywords

Scope

×

List

Grid

Name	Value		Scope
Prefix	Demo		Release
ServerName	DevServer		Dev
ServerName	TestServer		Test
Password	*****		Release

Predefined  
Variables

Pipeline  
variables

Stage  
Variables

Variable  
groups

Normal and  
Secret variables

# Understand variable groups

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

- 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

# Demonstration: Creating and managing variable groups

**DEMO**

# Module 06: Multi-stage YAML



# Describe Deployment Jobs strategies



Enable initialization

---



Deploy the update

---



Route traffic to the updated version

---



Test the updated version after routing traffic

---



If there's a failure, run steps to restore to the last known good version



# Describe lifecycle hooks



**preDeploy** – Used to run steps that initialize resources before application deployment starts.

---



**deploy** – Used to run steps that deploy your application.

---



**routeTraffic** – Used to run steps that serve the traffic to the updated version.

---



**postRouteTraffic** – Used to run the steps after the traffic is routed.

---



**on: failure or on: success** – Used to run steps for rollback actions or clean-up.

# Describe Deployment Jobs strategies – RunOnce

**runOnce** is the simplest deployment strategy wherein all the lifecycle hooks are executed:

- preDeploy
- deploy
- routeTraffic
- postRouteTraffic

Then, either **on: success** or **on: failure** is executed.

```
strategy:
  runOnce:
    preDeploy:
      pool: [ server | pool ] # See pool schema.
      steps:
        - script: [ script | bash | pwsh | powershell |
checkout | task | templateReference ]
    deploy:
      pool: [ server | pool ]
      steps:
        . . .
    routeTraffic:
      pool: [ server | pool ]
      steps:
        . . .
    postRouteTraffic:
      pool: [ server | pool ]
      steps:
        . . .
    on:
      failure:
        pool: [ server | pool ]
        steps:
          . . .
      success:
        pool: [ server | pool ]
        steps:
          . . .
```

# Describe Deployment Jobs strategies – Rolling

- A rolling deployment replaces instances of the previous version of an application with instances of the new version.
- It can be configured by specifying the keyword `rolling:` under the `strategy:` node.

```
strategy:
  rolling:
    maxParallel: [ number or percentage as x% ]
    preDeploy:
      steps: [ script | bash | pwsh | powershell |
checkout | task | templateReference ]
    deploy:
      steps:
        . . .
    routeTraffic:
      steps:
        . . .
    postRouteTraffic:
      steps:
        . . .
    on:
      failure:
        steps:
          . . .
      success:
        steps:
          . . .
```

# Describe Deployment Jobs strategies – Canary

- Advanced deployment strategy that helps mitigate the risk involved in rolling out new versions of applications.
- Roll out the changes to a small subset of servers first.
- Release it to more servers in your infrastructure and route more traffic to it.

```
strategy:
  canary:
    increments: [ number ]
    preDeploy:
      pool: [ server | pool ] # See pool schema.
      steps:
        - script: [ script | bash | pwsh | powershell | checkout
          | task | templateReference ]
    deploy:
      pool: [ server | pool ]
      steps:
        . . .
    routeTraffic:
      pool: [ server | pool ]
      steps:
        . . .
    postRouteTraffic:
      pool: [ server | pool ]
      steps:
        . . .
    on:
      failure:
        pool: [ server | pool ]
        steps:
          . . .
      success:
        pool: [ server | pool ]
        steps:
          . . .
```

# Describe Deployment Jobs strategies – Canary for AKS example

- In the following example, the canary strategy for AKS will first deploy the changes with 10 percent pods, followed by 20 percent, while monitoring the health during postRouteTraffic. If all goes well, it will promote to 100 percent.

```
jobs:
- deployment:
  environment: smarthotel-dev.bookings
  pool:
    name: smarthotel-devPool
  strategy:
    canary:
      increments: [ 10,20 ]
      preDeploy
      steps:
        - script: initialize, cleanup...
      deploy:
      steps:
        - script: echo deploy updates...
        - task: KubernetesManifest@0
      inputs:
        actions: $(strategy.action)
        namespace: 'default'
        strategy: $(strategy.name)
        percentage: $(strategy.increment)
        manifest: 'manifest.yml'
      postRouteTraffic:
        pool: server
        steps:
          - script: echo monitor application health...
      on:
        failure:
          steps:
            - script: echo clean-up, rollback...
        success:
          steps:
            - script: echo checks passed, notify...
```

# Module 07: Automate inspection of health



# Automate inspection of health

Stay informed about your process and releases:

- Release gates
- Events, subscriptions, and notifications
- Service hooks
- Reporting

# Explore events and notifications

- Actions in Azure DevOps trigger events
- Users can subscribe to events and get notified receiving an email
- Notifications can be managed centrally and personally



# Explore 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				

# Demonstration: Set up service hooks to monitor the pipeline

**DEMO**

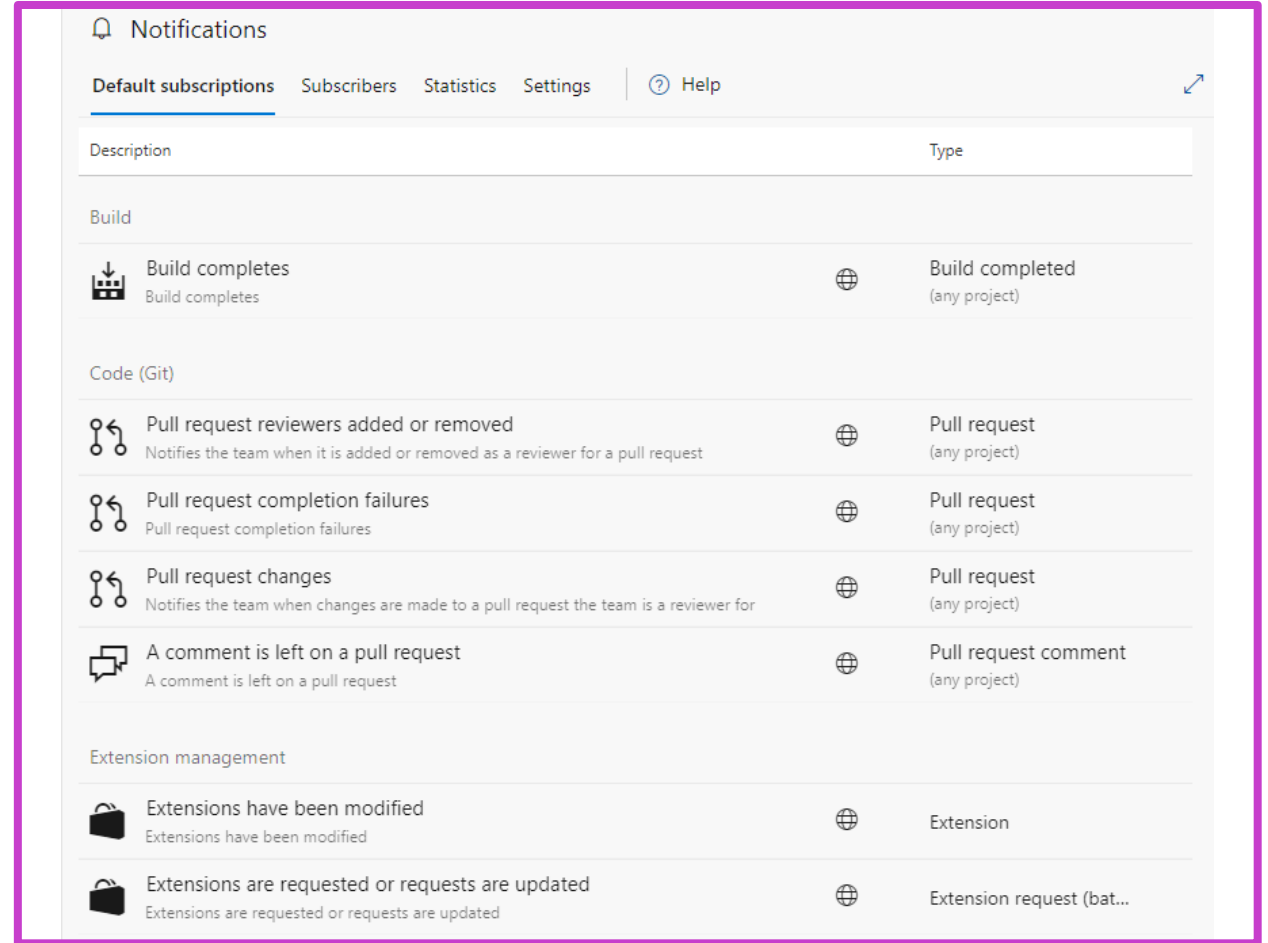
# Configure Azure DevOps notifications

You can get notified when changes occur to the following items:








- Work items
- Code reviews
- Pull requests
- Source control files (TFVC or Git)
- Builds
- Release

Subscriptions arise from the following instances:


- Out-of-the-box (OOB) or default.
- Created by an administrator for a team or group that you belong to.
- Created by you.



The screenshot shows the 'Notifications' page in Azure DevOps. It has a navigation bar with 'Default subscriptions', 'Subscribers', 'Statistics', 'Settings', and 'Help'. Below the navigation bar is a table with two columns: 'Description' and 'Type'. The table is divided into sections: 'Build', 'Code (Git)', and 'Extension management'. Each section contains one or more rows of notifications with icons, descriptions, and types.

Description	Type
<strong>Build</strong>	
 Build completes Build completes	Build completed (any project)
<strong>Code (Git)</strong>	
 Pull request reviewers added or removed Notifies the team when it is added or removed as a reviewer for a pull request	Pull request (any project)
 Pull request completion failures Pull request completion failures	Pull request (any project)
 Pull request changes Notifies the team when changes are made to a pull request the team is a reviewer for	Pull request (any project)
 A comment is left on a pull request A comment is left on a pull request	Pull request comment (any project)
<strong>Extension management</strong>	
 Extensions have been modified Extensions have been modified	Extension
 Extensions are requested or requests are updated Extensions are requested or requests are updated	Extension request (bat...

# Demonstration: Configure Azure DevOps notifications

 Azure DevOps

Organization Settings

Search Settings

General

Overview

Projects

Users

Billing

Auditing

Global notifications

Usage

Extensions











Azure Active Directory

Security

Settings / Global notifications

Notifications

Default subscriptionsSubscribersStatisticsSettingsHelp

Description	Type
Build	
 Build completes Build completes	 Build completed (any project)
Code (Git)	
 Pull request reviewers added or removed Notifies the team when it is added or removed as a reviewer for a pull request	 Pull request (any project)
 Pull request completion failures Pull request completion failures	 Pull request (any project)
 Pull request changes Notifies the team when changes are made to a pull request the team is a reviewer for	 Pull request (any project)
 A comment is left on a pull request A comment is left on a pull request	 Pull request comment (any project)

DEMO

# Configure GitHub notifications

By default, you automatically watch all repositories you create subscribe to conversations when you have:

- Not disabled automatic watching for repositories or teams you've joined in your notification settings.
- Been assigned to an issue or pull request.
- Opened a pull request, issue, or created a team discussion post.
- Commented on a thread.
- Subscribed to a thread manually by clicking Watch or Subscribe.
- Had your username @mentioned.
- Changed the thread's state by closing an issue or merging a pull request.
- Had a team you're a member of @mentioned.

## Notifications

Choose how you receive notifications. These notification settings apply to the [things you're watching](#).

### Automatic watching

When you're given push access to a repository, automatically receive notifications for it.

☐ Automatically watch repositories

When you're added to or join a team, automatically receive notifications for that team's discussions.

☒ Automatically watch teams

### Participating

Notifications for the conversations you are participating in, or if someone cites you with an @mention.

☒ Email

☒ Web and Mobile

### Watching

Notifications for all repositories, teams, or conversations you're watching.

☒ Email

☒ Web and Mobile

# Demonstration: Configure GitHub notifications

## Notifications

Choose how you receive notifications. These notification settings apply to the [things you're watching](#).

### Automatic watching

When you're given push access to a repository, automatically receive notifications for it.

☐ Automatically watch repositories

When you're added to or join a team, automatically receive notifications for that team's discussions.

☒ Automatically watch teams

### Participating

Notifications for the conversations you are participating in, or if someone cites you with an @mention.

☒ Email

☒ Web and Mobile

### Watching

Notifications for all repositories, teams, or conversations you're watching.

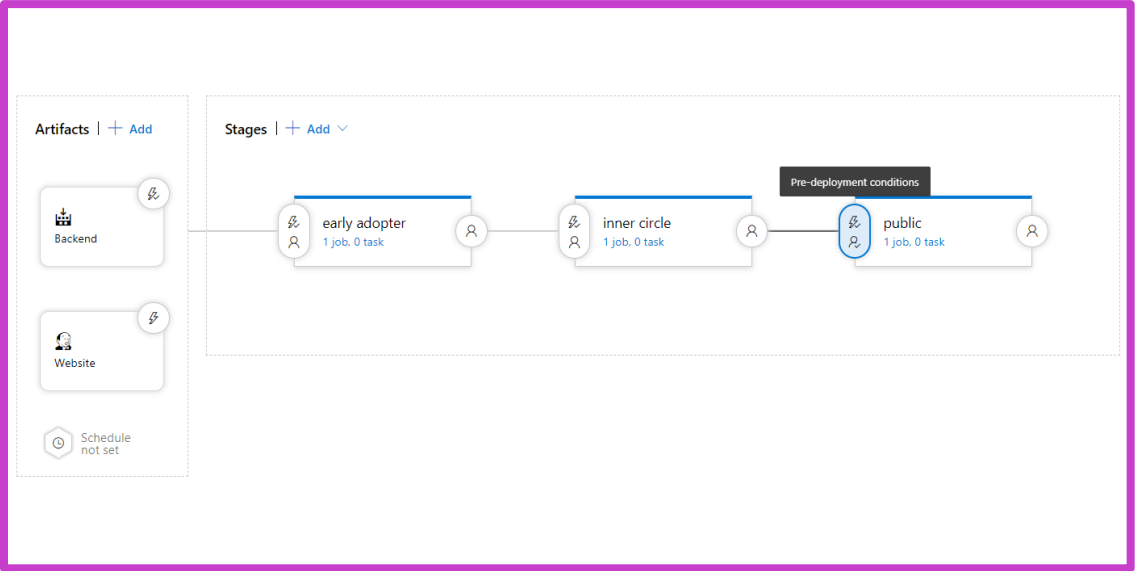
DEMO

# Explore how to measure quality of your release process

Release Branch Runs - Default

Environments

Sps.SelfTest	✓	100%	✓	100%	✓	100%	✓	100%	✓	100%	✓	100%	▶
Sps.SelfHost	✓	100%	✓	100%	✓	100%	✓	100%	✓	100%	▶	▶	
Tfs.SelfHost Set 1	✓	100%	✓	100%	✓	100%	✓	100%	✓	100%	▶	▶	
Tfs.SelfHost Set 2	✓	100%	✓	100%	✓	100%	✗	98.69%	✓	100%	▶	▶	
Tfs.SelfTest	✓	100%	✓	100%	✓	100%	✓	100%	✓	100%	▶	▶	
Tfs.Deploy			✓	100%	✓	100%			✓	100%	▶	▶	
TfsOnPrem.SelfHost	✓	100%	✓	100%	✓	100%	✓	100%	✓	100%	▶	▶	
TfsOnPrem.SelfTest	✓	100%	✓	100%	✓	100%	✓	100%	✓	100%	▶	▶	



Visualize your release process

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

Dashboard widgets

# Examine 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 Microsoft Azure DevOps work item for Feature 344, titled "344 Shopping Cart should be personalized". The work item is assigned to "Unassigned" and has 0 comments. It is categorized as "New" with the area "Test demo" and reason "New feature". The description states "The shopping cart should know the user". The acceptance criteria section is empty, with a link to add criteria. The "Release Notes" section is highlighted with a red box and contains the text "Here can be the commercial release notes." The discussion section is empty, with a prompt to add a comment. The right sidebar shows the status "New", start and target dates, priority "2", effort, business value, time criticality, and value area "Business". The development section shows a link to add a link and a note that development hasn't started on this item. The related work section shows a link to add a link and a note that there are no links in this group.

FEATURE 344

344 Shopping Cart should be personalized

Unassigned 0 comments Add tag

Save Follow Refresh Undo More

Updated by rvanosnabrugge just now

Details History Links Attachments

**Description**

The shopping cart should know the user

**Acceptance Criteria**

Click to add Acceptance Criteria

**Release Notes**

Here can be the commercial release notes.

**Discussion**

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

**Status**

Start Date

Target Date

**Details**

Priority 2

Effort

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.



# Examine considerations for choosing release management tools

- Artifacts and artifact source
- Triggers and schedules
- Approvals and gates
- Stages
- Build and release tasks
- Traceability, auditability and security

# Explore common release management tools

GitHub Actions



Azure Pipelines



Jenkins



Circle CI



GitLab Pipelines



Atlassian Bamboo



# Labs



# Lab: Controlling deployments using Release Gates



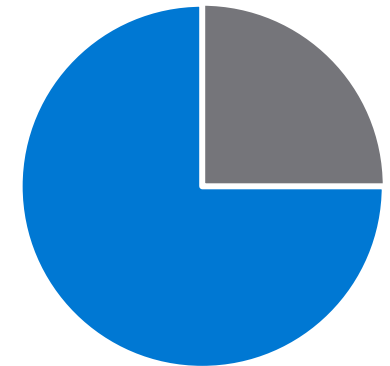
## Lab overview:

This lab covers the configuration of the deployment gates and details how to use them to control execution of Azure pipelines.

## Objectives:

- Configure release pipelines
- Configure release gates
- Test release gates

## Duration:



# Learning Path review and takeaways



# What did you learn?

- 1 Explain the terminology used in Azure DevOps and other release management tooling
- 2 Describe what a build and release task is, what it can do, and some available deployment tasks
- 3 Explain why you sometimes need multiple release jobs in one release pipeline
- 4 Differentiate between a release and a deployment
- 5 Define the components of a release pipeline
- 6 Use release variables and stage variables in your release pipeline
- 7 Deploy to environment securely using a service connection
- 8 List the different ways to inspect the health of your pipeline and release by using alerts, service hooks, and reports
- 9 Explain things to consider when designing your release strategy

# Learning Path 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?
- 4 How can you identify a default release variable?
- 5 What can you use to prevent a deployment in Azure DevOps when a security testing tool finds a compliance problem?
- 6 Even if you create exactly what a user requested at the start of the project, the solution will often be unsuitable for the same user. Why?

