

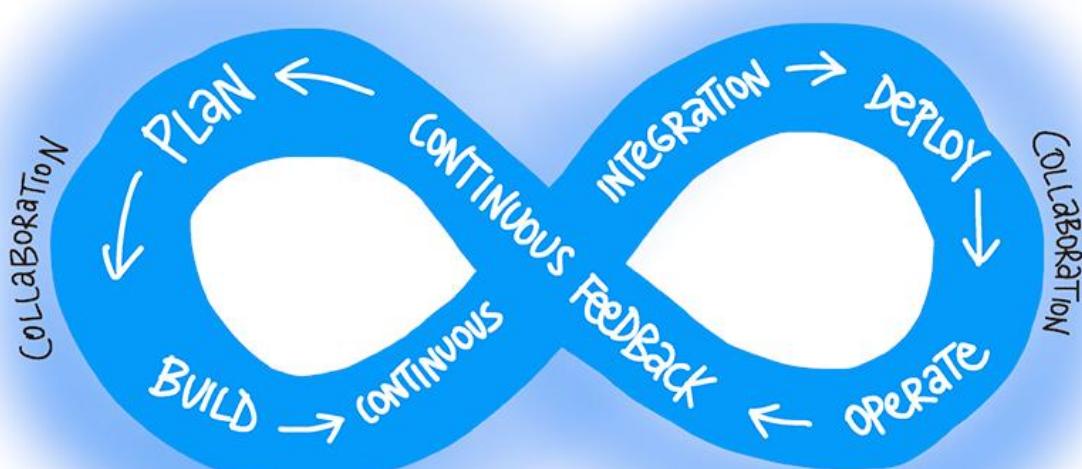
# AZ-400: Designing and Implementing Microsoft DevOps Solutions

## What is DevOps?

Completed 100 XP

- 3 minutes

The contraction of "Dev" and "Ops" refers to replacing siloed Development and Operations. The idea is to create multidisciplinary teams that now work together with shared and efficient practices and tools. Essential DevOps practices include agile planning, continuous integration, continuous delivery, and monitoring of applications. DevOps is a constant journey.



## Understand your cycle time

Let us start with a basic assumption about software development. We will describe it with the **OODA (Observe, Orient, Decide, Act) loop**. Originally designed to keep fighter pilots from being shot out of the sky, the OODA loop is an excellent way to think about

staying ahead of your competitors. You start with observing business, market, needs, current user behavior, and available telemetry data. Then you orient with the enumeration of options for what you can deliver, perhaps with experiments. Next, you decide what to pursue, and you act by delivering working software to real users. You can see all occurring in some cycle time.

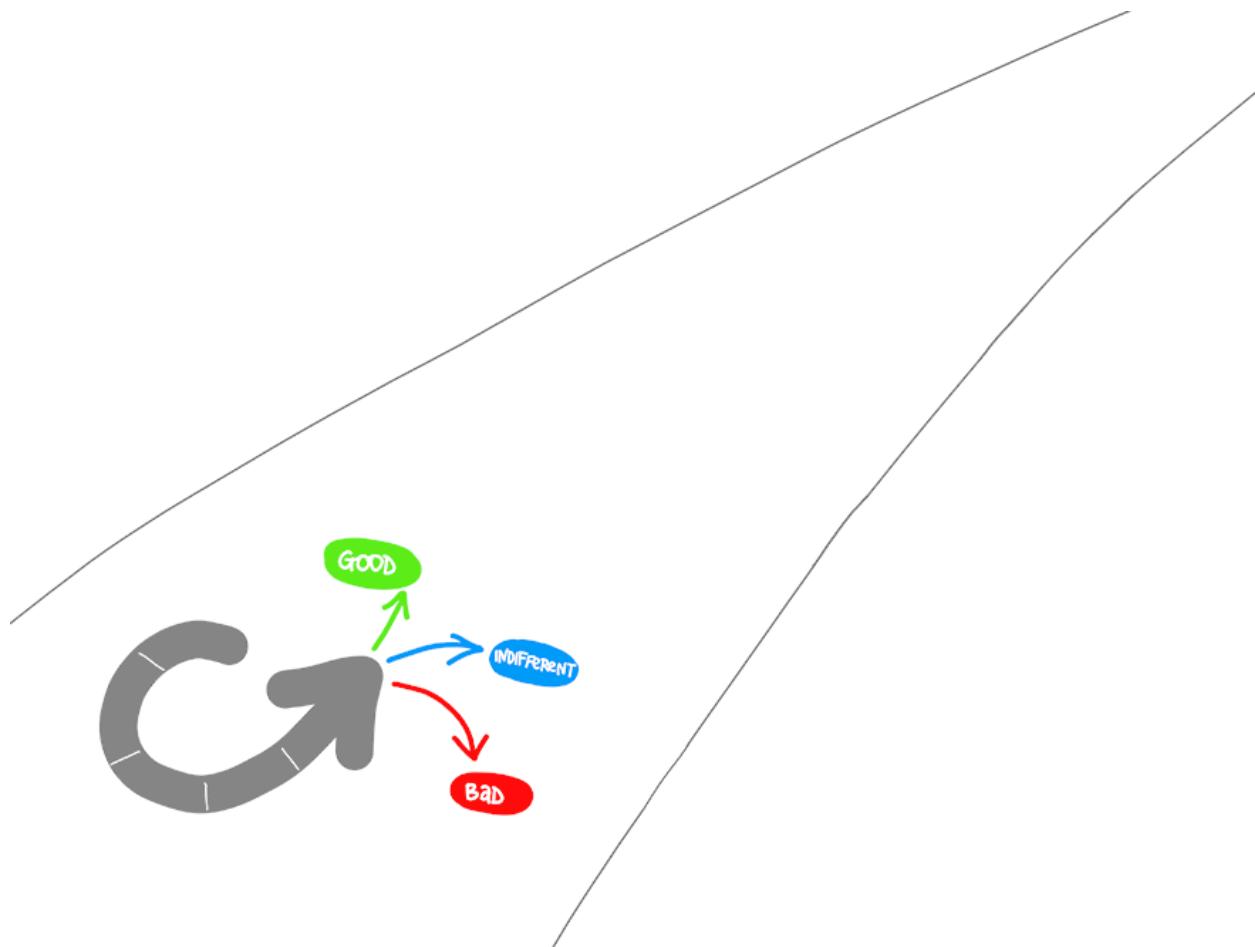


## Become data-informed

We recommend you use data to inform what to do in your next cycle. Many experience reports tell us that roughly one-third of the deployments will have negative business results. Approximately one-third will have positive results, and one-third will make no difference. Fail fast on effects that do not advance the business and double down on outcomes that support the business. Sometimes the approach is called pivot or persevere.

## Strive for validated learning

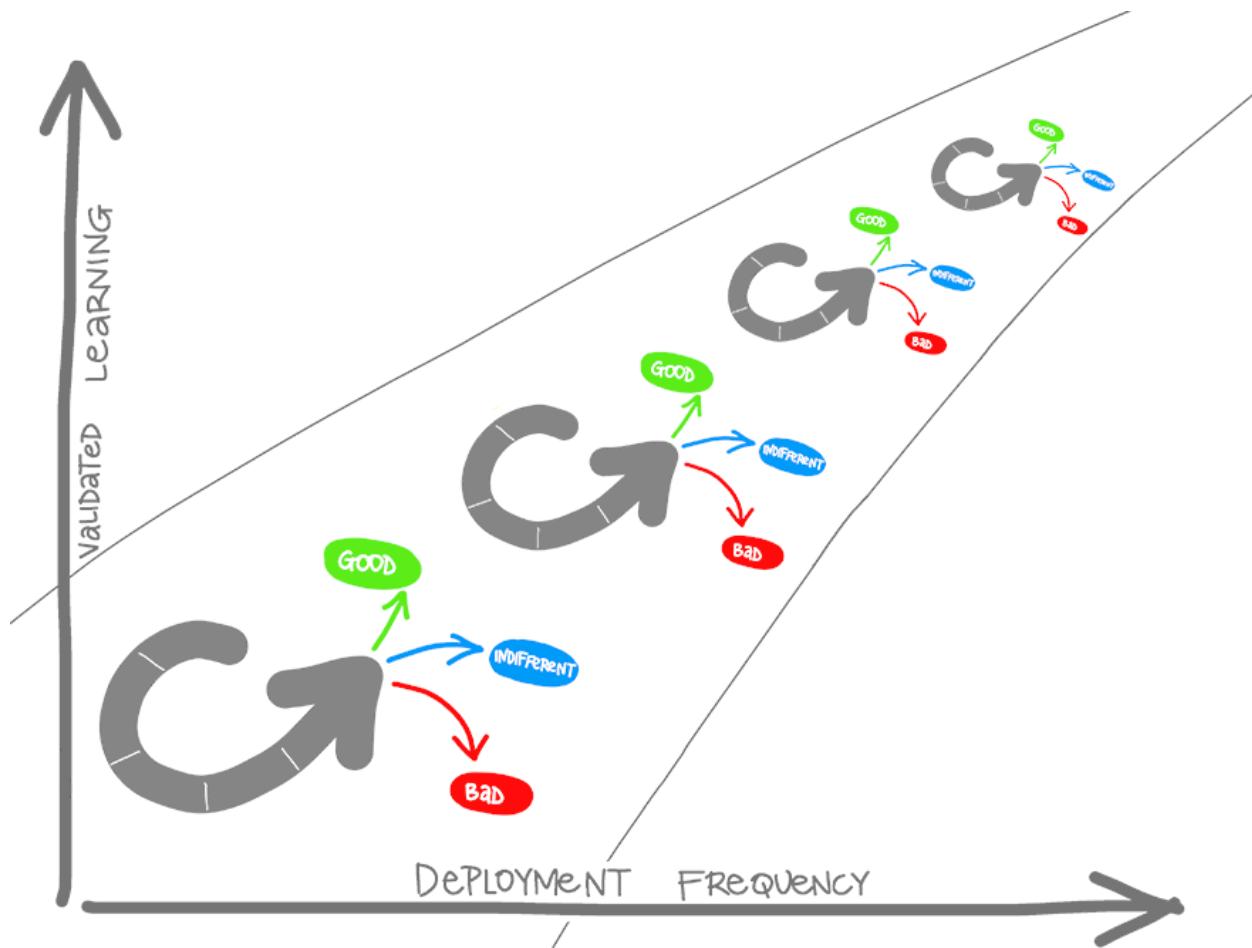
How quickly you can fail fast or double down is determined by your cycle time. Also, in how long that loop takes, or in lean terms. Your cycle time determines how quickly you can gather feedback to determine what happens in the next loop. The feedback that you collect with each cycle should be factual, actionable data. We call it validated learning.



## Shorten your cycle time

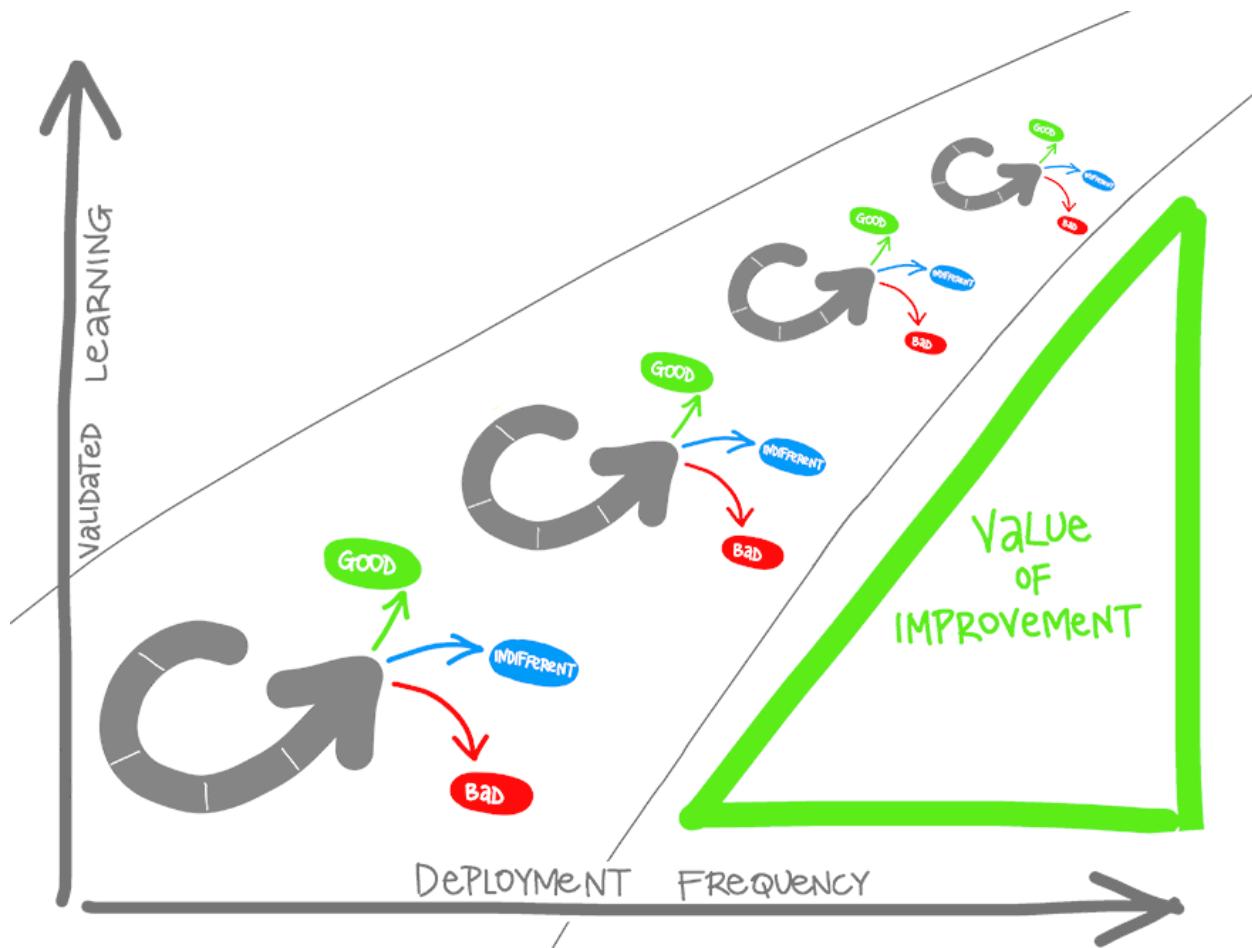
When you adopt DevOps practices:

- You shorten your cycle time by working in smaller batches.
- Using more automation.
- Hardening your release pipeline.
- Improving your telemetry.
- Deploying more frequently.



## Optimize validated learning

The more frequently you deploy, the more you can experiment. The more opportunity you have to pivot or persevere and gain validated learning each cycle. This acceleration in validated learning is the value of the improvement. Think of it as the sum of progress that you achieve and the failures that you avoid.



## Explore the DevOps journey

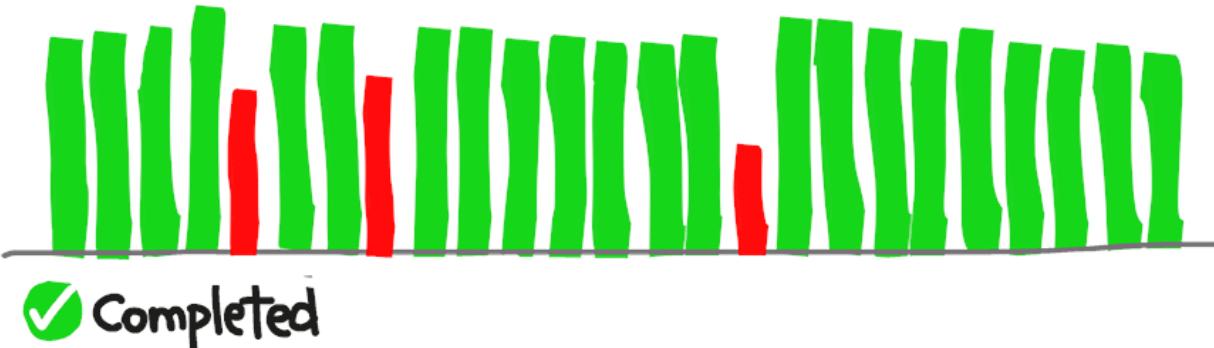
Completed 100 XP

- 4 minutes

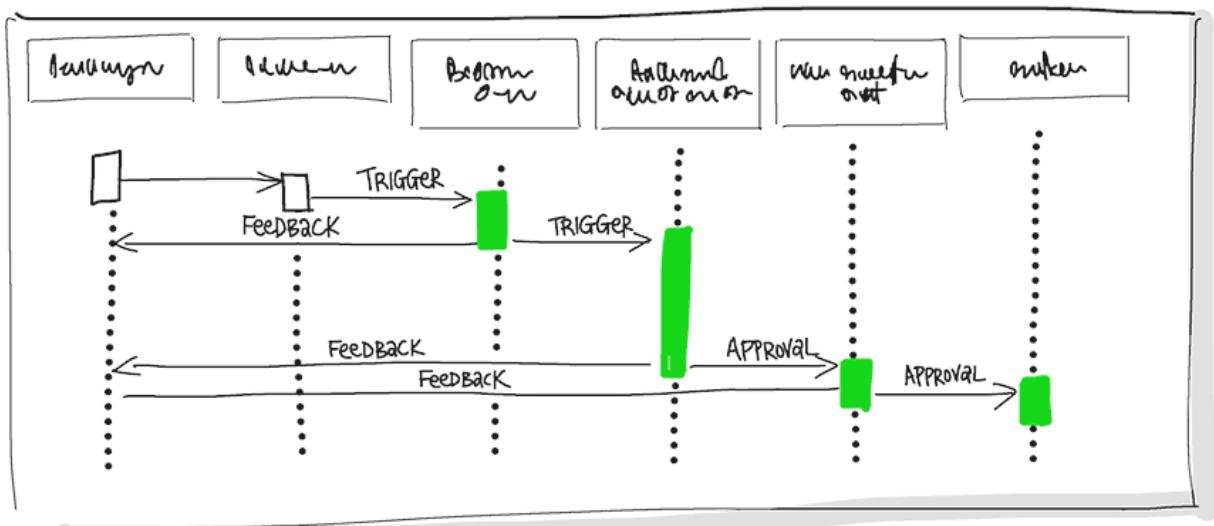
Remember, the goal is to shorten cycle time. Start with the release pipeline. How long does it take to deploy a change of one line of code or configuration? Ultimately, that is the brake on your velocity.

- Continuous Integration drives the ongoing merging and testing of code, leading to an early finding of defects. Other benefits include less time wasted fighting merge issues and rapid feedback for development teams.

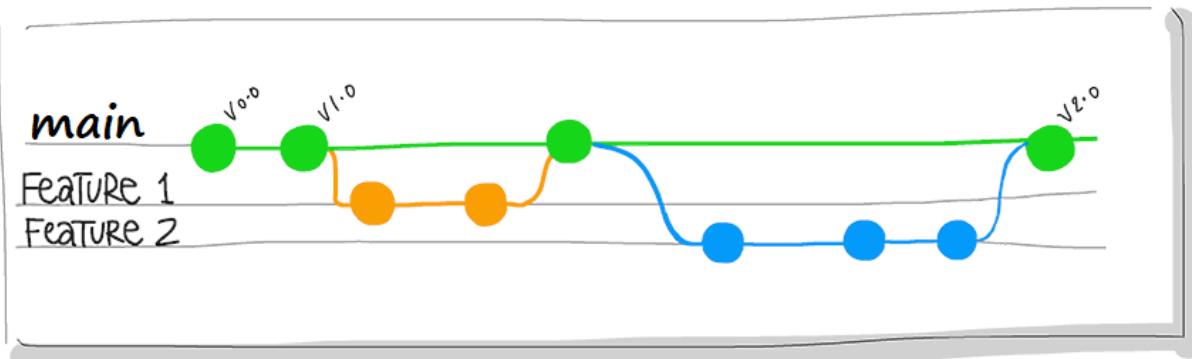
# BUILD Succeeded



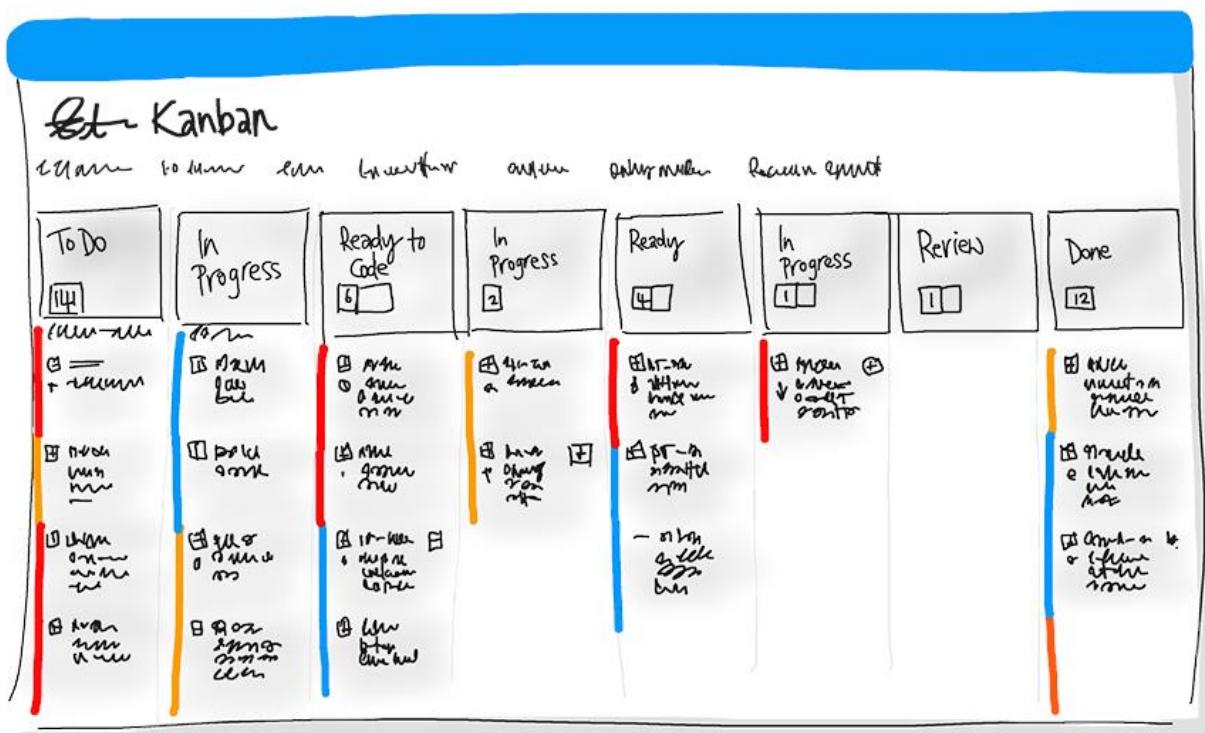
- Continuous Delivery of software solutions to production and testing environments helps organizations quickly fix bugs and respond to ever-changing business requirements.



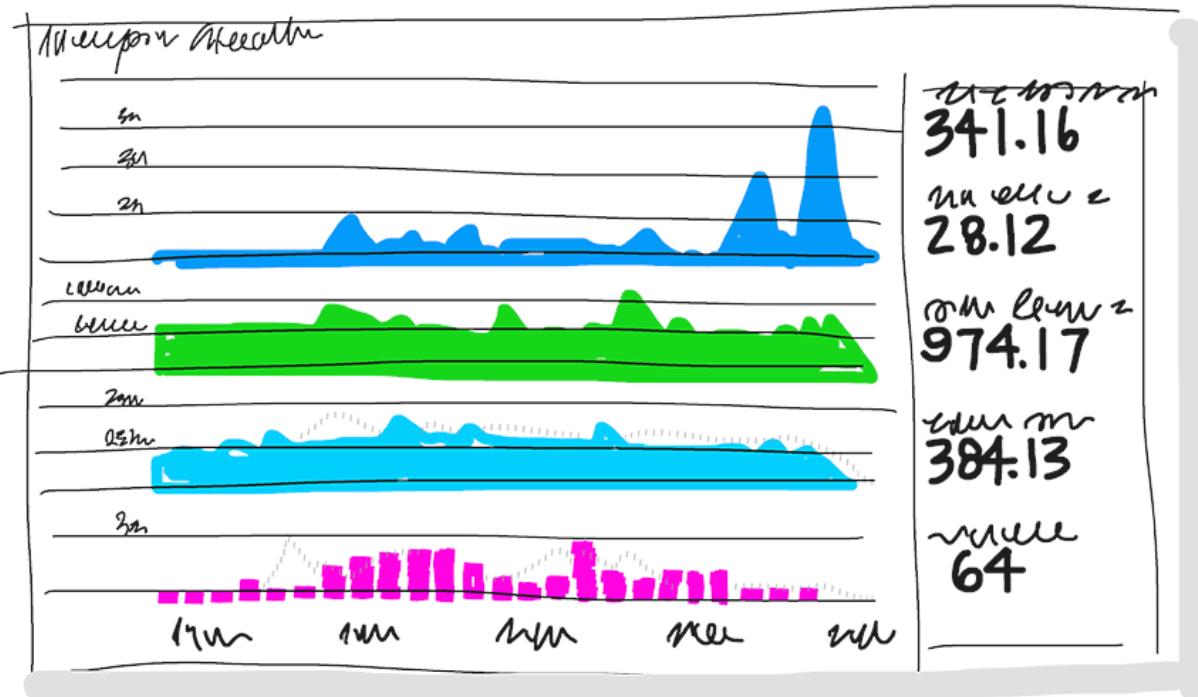
- Version Control, usually with a Git-based Repository, enables teams worldwide to communicate effectively during daily development activities. Also, integrate with software development tools for monitoring activities such as deployments.



- Use Agile planning and lean project management techniques to:
  - Plan and isolate work into sprints.
  - Manage team capacity and help teams quickly adapt to changing business needs.
  - A DevOps Definition of Done is working software collecting telemetry against the intended business goals.



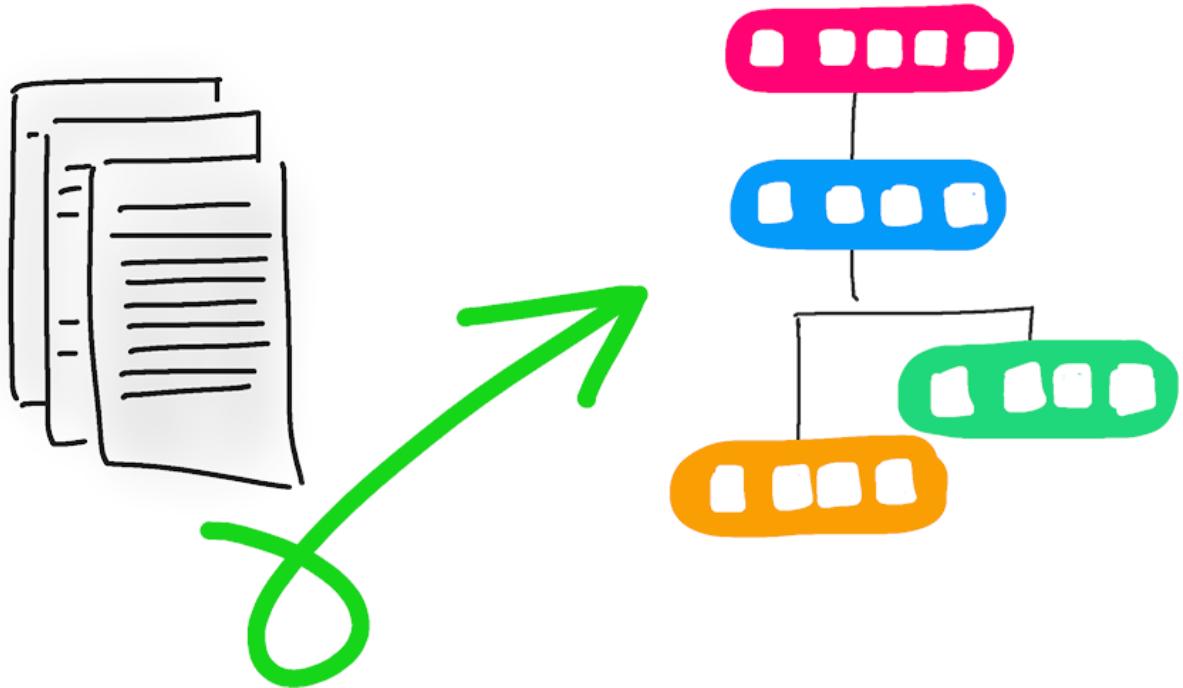
- Monitoring and Logging of running applications. Including production environments for application health and customer usage. It helps organizations create a hypothesis and quickly validate or disprove strategies. Rich data is captured and stored in various logging formats.



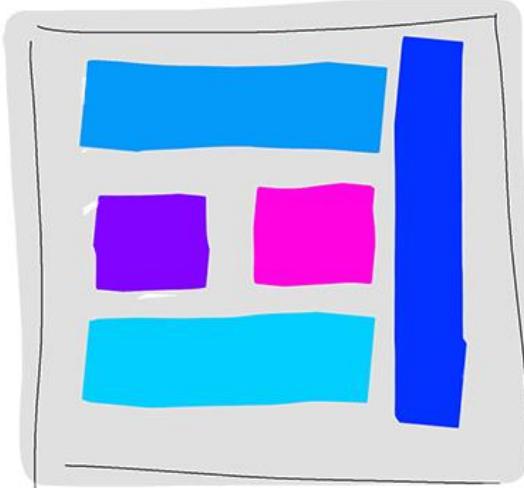
- Public and Hybrid Clouds have made the impossible easy. The cloud has removed traditional bottlenecks and helped commoditize Infrastructure. You can use Infrastructure as a Service (IaaS) to lift and shift your existing apps or Platform as a Service (PaaS) to gain unprecedented productivity. The cloud gives you a data center without limits.



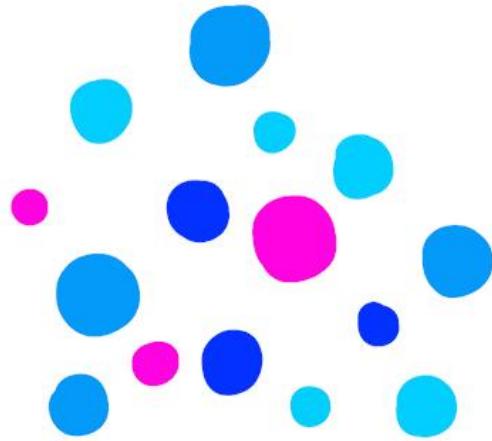
- Infrastructure as Code (IaC): Enables the automation and validation of the creation and teardown of environments to help deliver secure and stable application hosting platforms.



- Use Microservices architecture to isolate business use cases into small reusable services that communicate via interface contracts. This architecture enables scalability and efficiency.

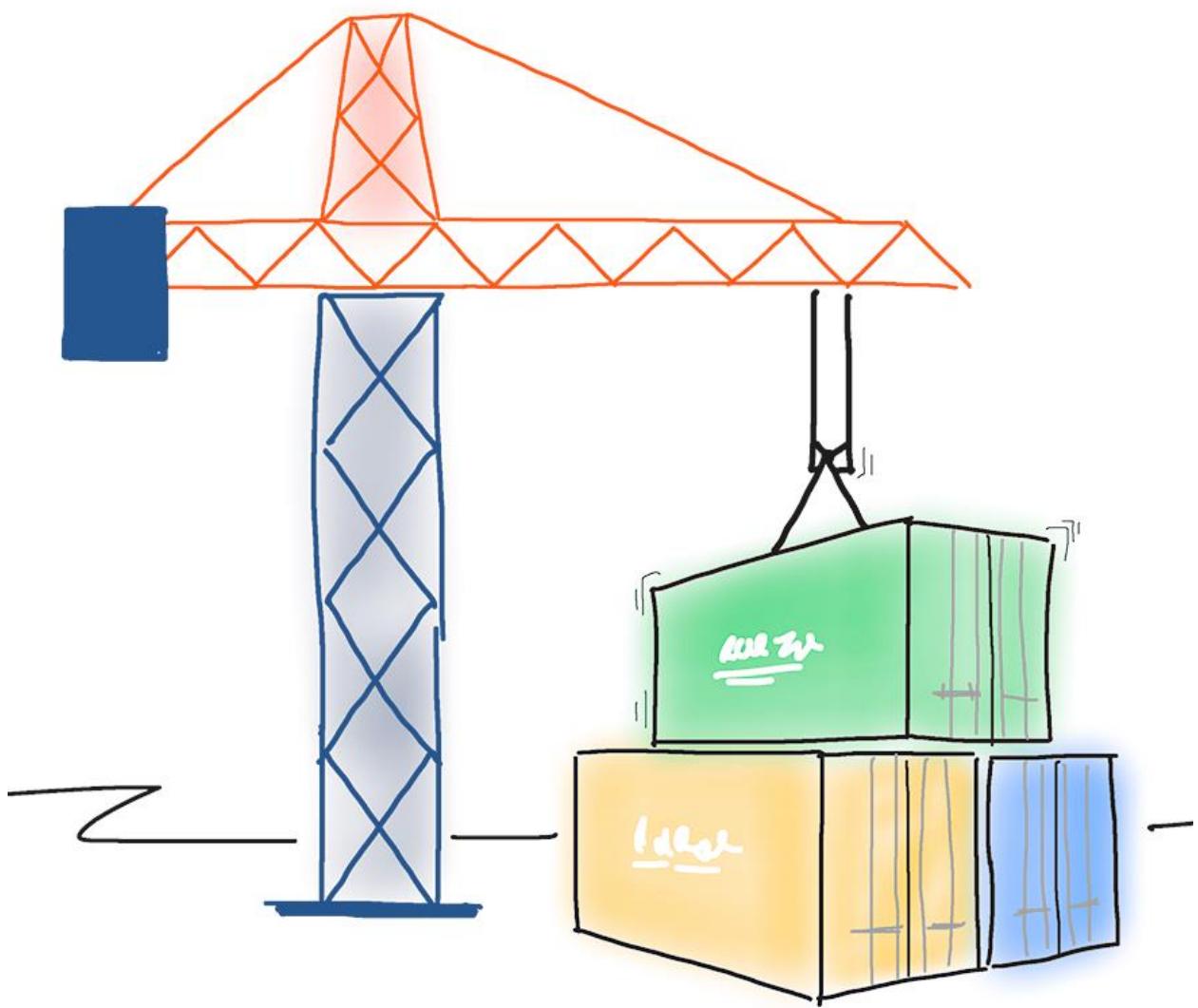


MONOLITHIC/LAYERED



MICROSERVICES

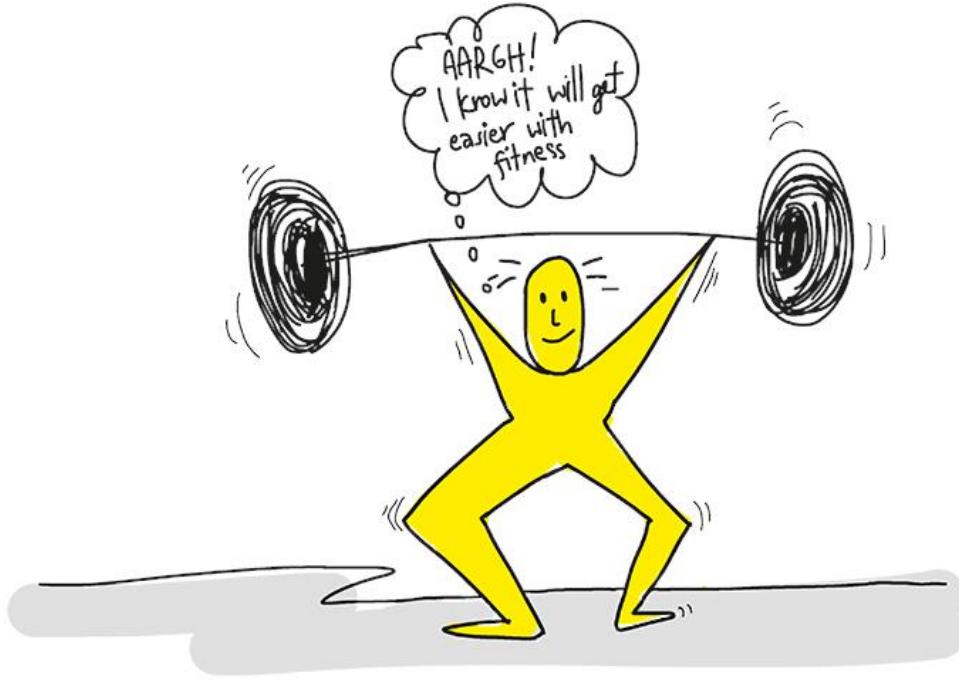
- Containers are the next evolution in virtualization. They're much more lightweight than virtual machines, allow much faster hydration, and easily configure files.



## DevOps may hurt at first.

If it hurts, do it more often. Adopting new practices like going to the gym is likely to hurt first. The more you exercise the new techniques, the easier they'll become.

Like training at the gym, where you first exercise large muscles before small muscles, adopt practices that have the most significant impact. Cross-train to develop synergy.



## Identify transformation teams

Completed 100 XP

- 2 minutes

Unless you're building an entirely new organization, one of the significant challenges with any DevOps Transformation Project is taking actions that conflict. At least in some way with ongoing business states.

The first challenge is the availability of staff. Suppose the staff members leading the transformation project are also involved in existing day-to-day work within the organization.

It will be challenging to focus on the transformation when their current role directly impacts customer outcomes.

We all know desperate situations involving customers will always win over a long-term project, like DevOps Transformations.

Another issue will be how the organization operates—implementing existing processes and procedures to support current business outcomes.

The disruption required for a true DevOps Transformation will usually challenge existing processes and procedures. Doing that is often difficult.

In the book "Beyond the Idea: How to Execute Innovation," Dr. Vijay Govindarajan and Dr. Chris Trimble noted when successful, they have researched what is involved in allowing Innovation to occur in organizations.

It has often been despite the existing organizational processes. Concluding it only works by creating a separate team to pursue the transformation.

For DevOps transformations, the separate team should be composed of staff members. Focused on and measured the transformation outcomes and not involved in the day-to-day operational work. The team might also include external experts that can fill the knowledge gaps—also helping to advise on processes that are new to the existing staff members.

Ideally, the staff members recruited should already be well-regarded throughout the organization. They should offer a broad knowledge base to think outside the box as a group.

## Explore shared goals and define timelines

Completed 100 XP

- 2 minutes

### Explore shared goals

These outcomes should include specific, measurable targets like:

- Reduce the time spent on fixing bugs by 60%.
- Reduce the time spent on unplanned work by 70%.
- Reduce the out-of-hours work required by staff to no more than 10% of total working time.
- Remove all direct patching of production systems.

#### Note

One of the aims of DevOps is to provide more excellent customer value, so outcomes should have a customer value focus.

## Define timelines for goals

Measurable goals also need to have timelines. While it is easy to set longer-term goals, it is also easy to put off work when you do not require it for a while.

It is essential to have an ongoing series of short-term goals. Overall, projects should have timelines that span anywhere from a few months to a year or two in any DevOps transformation project.

Every few weeks, the improvements should be clear and measurable. Ideally, evident to the organization or its customers.

The timeline should not be too short and should always be challenging yet achievable. A review should occur after each short-term goal to help plan the next one.

There are several advantages of the shorter timelines:

- It is easier to change plans or priorities when necessary.
- The reduced delay between doing work and getting feedback helps ensure that the learnings and feedback are incorporated quickly.
- It is easier to keep organizational support when positive outcomes are clear.

# Choose the right project

## Explore greenfield and brownfield projects

Completed 100 XP

- 2 minutes

The terms greenfield and brownfield have their origins in residential and industrial building projects.

A greenfield project is one done on a green field, undeveloped land. A brownfield project is done on the used ground for other purposes.

Because of the land use that has once occurred, there could be challenges reusing the land. Like existing buildings, some would be obvious but less obvious, like polluted soil.

## Applied to software or DevOps Projects

The same terms are routinely applied to software projects and commonly describe DevOps Projects.

On the surface, it can seem that a greenfield DevOps project would be easier to manage and to achieve success.

- There was no existing codebase.
- No existing team dynamics or politics. Possibly no current, rigid processes.

A common misconception is that DevOps is only for greenfield projects and suits startups best. However, DevOps can also succeed with brownfield projects.

The beauty of these projects is that there's often a large gap between customer expectations and delivery.

The teams involved may well realize that the status quo needs to change. They've lived the challenges and the limitations associated with what they're currently doing.

# Decide when to use greenfield and brownfield projects

Completed 100 XP

- 3 minutes

When starting a DevOps transformation, you might need to choose between greenfield and brownfield projects.

There's a common misconception that we need to demystify that DevOps suits greenfield projects better than brownfield projects.

## Greenfield projects

A greenfield project will always appear to be a more accessible starting point. A blank slate offers the chance to implement everything the way that you want.

You might also have a better chance of avoiding existing business processes that don't align with your project plans.

Suppose current IT policies don't allow the use of cloud-based infrastructure. In that case, the project might be qualified for entirely new applications designed for that environment from scratch.

For example, you can sidestep internal political issues that are well entrenched.

## Brownfield projects

Usually, brownfield projects come with:

- The baggage of existing codebases.
- Existing teams.
- A significant amount of technical debt.

But, they can still be ideal projects for DevOps transformations.

When your teams spend large percentages of their time just maintaining existing brownfield applications, you have limited ability to work on new code.

It's essential to find a way to reduce that time and to make software release less risky. A DevOps transformation can provide that.

The limitations will have often worn down the existing team members. For example, they're working in the past and aren't keen to experiment with new ideas.

The system is often crucial for organizations. It might also be easier to gain more robust management buy-in for these projects because of the potential benefits delivered.

Management might also have a stronger sense of urgency to point brownfield projects in an appropriate direction when compared to greenfield projects that don't currently exist.

## Take the first step

Eventually, the goal will be to evolve your entire organization. In looking to take the first step, many organizations start with a greenfield project and then move on.

# Decide when to use systems of record versus systems of engagement

Completed 100 XP

- 2 minutes

When selecting systems as candidates for starting a DevOps transformation, it is necessary to consider the types of systems that you operate.

Some researchers suggest that organizations often use Bimodal IT, a practice of managing two separate, coherent modes of IT delivery - one focused on stability and predictability and the other on agility.

## Systems of record

Systems that provide the truth about data elements are often-called systems of record. These systems have historically evolved slowly and carefully. For example, it is crucial that a banking system accurately reflects your bank balance. Systems of record emphasize accuracy and security.

## Systems of engagement

Many organizations have other systems that are more exploratory. These often use experimentation to solve new problems. Systems of engagement are modified regularly. Usually, it is a priority to make quick changes over ensuring that the changes are correct.

There is a perception that DevOps suits systems of engagement more than systems of record. The lessons from high-performing companies show that is not the case.

Sometimes, the criticality of doing things right with a system of record is an excuse for not implementing DevOps practices.

Worse, given the way that applications are interconnected, an issue in a system of engagement might end up causing a problem in a system of record anyway.

Both types of systems are great. At the same time, it might be easier to start with a system of engagement when first beginning a DevOps Transformation.

DevOps practices apply to both types of systems. The most significant outcomes often come from transforming systems of record.

## Identify groups to minimize initial resistance

Completed 100 XP

- 2 minutes

Not all staff members within an organization will be receptive to the change required for a DevOps transformation.

In discussions around continuous delivery, we usually categorize users into three general buckets:

- **Canary** users voluntarily test bleeding edge features as soon as they're available.
- **Early adopters** who voluntarily preview releases, considered more refined than the code that exposes canary users.
- **Users** who consume the products after passing through canary and early adopters.

It's essential to find staff members keen to see new features as soon as they're available and highly tolerant of issues when choosing Canary.

Early adopters have similar characteristics to the Canaries. They often have work requirements that make them less tolerant of issues and interruptions to work.

While development and IT operations staff might generally be less conservative than users.

The staff will also range from traditional to early adopters, and others happy to work at the innovative edge.

## Ideal target improvements

It's also important to roll out changes incrementally. There is an old saying in the industry that any successful large IT system was previously a successful small IT system.

Large-scale systems rolled out all at once have an abysmal record of success. Most fail, no matter how much support management has provided.

When starting, it is essential to find an improvement goal that:

- It can be used to gain early wins.
- It is small enough to be achievable in a reasonable timeframe.
- Has benefits that are significant enough to be evident to the organization.

It allows constant learning from rapid feedback and recovering from mistakes quickly.

### Note

The aim is to build a snowball effect where each new successful outcome adds to previous successful results. It will maximize the buy-in from all affected.

## Identify project metrics and key performance indicators (KPIs)

Completed 100 XP

- 3 minutes

We spoke earlier about the importance of shared goals. It was also agreed upon by team members that the goals needed to be specific, measurable, and time-bound.

It is essential to establish (and agree upon) appropriate metrics and Key Performance Indicators (KPIs) to ensure these goals are measurable.

While there is no specific list of metrics and KPIs that apply to all DevOps Projects, the following are commonly used:

## Faster outcomes

- **Deployment Frequency.** Increasing the frequency of deployments is often a critical driver in DevOps Projects.
- **Deployment Speed.** It is necessary to reduce the time that they take.
- **Deployment Size.** How many features, stories, and bug fixes are being deployed each time?
- **Lead Time.** How long does it take from the creation of a work item until it is completed?

## Efficiency

- **Server to Admin Ratio.** Are the projects reducing the number of administrators required for a given number of servers?
- **Staff Member to Customers Ratio.** Is it possible for fewer staff members to serve a given number of customers?
- **Application Usage.** How busy is the application?
- **Application Performance.** Is the application performance improving or dropping? (Based upon application metrics)?

## Quality and security

- **Deployment failure rates.** How often do deployments (or applications) fail?
- **Application failure rates.** How often do application failures occur, such as configuration failures, performance timeouts, and so on?
- **Mean time to recover.** How quickly can you recover from a failure?
- **Bug report rates.** You do not want customers finding bugs in your code. Is the amount they are seeing increasing or lowering?
- **Test pass rates.** How well is your automated testing working?

- **Defect escape rate.** What percentage of defects are being found in production?
- **Availability.** What percentage of time is the application truly available for customers?
- **Service level agreement achievement.** Are you meeting your service level agreements (SLAs)?
- **Mean time to detection.** If there is a failure, how long does it take for it to be detected?

## Culture

- **Employee morale.** Are employees happy with the transformation and where the organization is heading? Are they still willing to respond to further changes? This metric can be challenging to measure but is often done by periodic, anonymous employee surveys.
- **Retention rates.** Is the organization losing staff?

### Note

It is crucial to choose metrics that focus on specific business outcomes and achieve a return on investment and increased business value.

# Describe team structures

## Explore agile development practices

Completed 100 XP

- 3 minutes

### Waterfall

Traditional software development practices involve:

- Determining a problem.
- Analyzing the requirements.
- Building and testing the required code.
- The delivery outcome to users.

Usually, all refer to as a waterfall approach.

The waterfall model follows a sequential order. A project development team only moves to the next development phase or testing if the previous step is completed successfully.

It's what an engineer would do when building a bridge or a building. So, it might seem appropriate for software projects as well.

However, the waterfall methodology has some drawbacks. One relates to the customer requirements.

For example, It doesn't matter if the customer requirements are defined accurately at the start of a project.

Usually, the project takes a long time, and the outcome may no longer match the customer's needs.

There's a real challenge with gathering customer requirements in the first place.

Taking a long time to deliver something would often be different from what the customer needs, even if you built exactly what the customer asked.

Customers often don't know what they want until they see it or can't explain what they need.

# Agile

By comparison, Agile methodology constantly emphasizes adaptive planning and early delivery with continual improvement.

Rather than restricting development to rigid specifications, it encourages rapid and flexible responses to changes as they occur.

In 2001, highly regarded developers published a manifesto for Agile software development.

They said that:

- Development needs to favor individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Respond to changes over following a plan.

Agile software development methods are based on releases and iterations:

- One release might consist of several iterations.
- Each iteration is like a small independent project.
- After being estimated and prioritization:
  - Features, bug fixes, enhancements, and refactoring width are assigned to a release.
  - And then assigned again to a specific iteration within the release, generally on a priority basis.
- At the end of each iteration, there should be tested working code.
- In each iteration, the team must focus on the outcomes of the previous iteration and learn from them.

Having teams focused on shorter-term outcomes is that teams are also less likely to waste time over-engineering features. Or allowing unnecessary scope creep to occur.

Agile software development helps teams keep focused on business outcomes.

## Comparison of the waterfall and agile methodologies

Expand table

## Waterfall

Divided into distinct phases.

It can be rigid.

All project development phases, such as design, development, and test, are completed once.

Define requirements at the start of the project with little change expected.

Focus on completing the project.

## Agile

Separates the project development lifecycle into smaller cycles. Known for flexibility.

It follows an iterative development approach where requirements may appear more than once.

Requirements are expected to change and evolve over time.

Focus on meeting customers' demands.

# Explore principles of agile development

Completed 100 XP

- 2 minutes

The [Agile Alliance](#) says its mission is to support people who explore and apply agile values and principles. Also, practices to make building software solutions more effective, humane, and sustainable.

They have published a [Manifesto for Agile Software Development](#).

And from the publication, they have distilled the [12 Principles Behind the Agile Manifesto](#).

- Our highest priority is to satisfy the customer through the early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of months to a couple of weeks, with a preference for a shorter timescale.
- Businesspeople and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work not done - is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- The team regularly reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

## Define organization structure for agile practices

Completed 100 XP

- 3 minutes

For most organizations, reorganizing to be agile is difficult. It requires a mind-shift and a culture-shift that challenges many existing policies and processes within the organization.

Good governance in organizations, particularly in large organizations, often leads to many relatively rigid rules, operating structures, and methods. It also tends to avoid a broad delegation of authority.

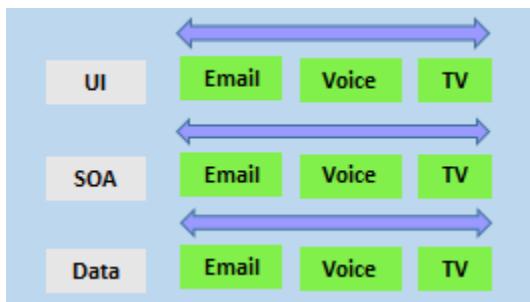
While most large organizations have not moved to an agile structure, most are now experimenting with doing so.

Their business environments are volatile and complex, and they have seen the limitations of their current systems, mainly an inability to cope with change quickly.

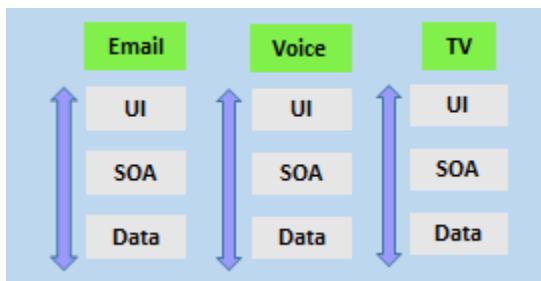
They realize that it is common today for long-term established businesses and their industries to be disrupted by startups.

### Horizontal vs. vertical teams

Traditionally, horizontal team structures divide teams according to the software architecture. In this example, the teams have been divided into the user interface, service-oriented architecture, and data teams:

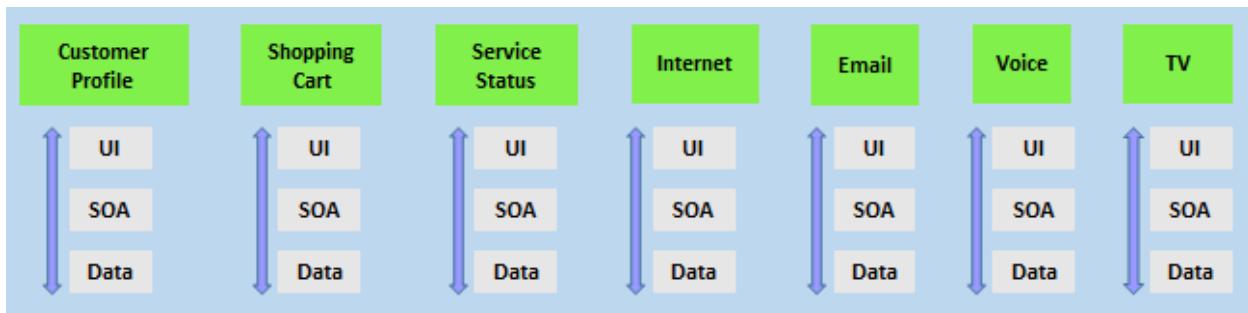


By comparison, vertical team structures span the architecture and are aligned with skillsets or disciplines:



Vertical teams have been shown to provide more good outcomes in Agile projects. Each product must have an identified owner.

Another key benefit of the vertical team structure is that scaling can occur by adding teams. In this example, feature teams have been created rather than just project teams:



## Explore ideal DevOps team members

Completed 100 XP

- 2 minutes

For a successful DevOps transformation, the aim is to find team members with the following characteristics:

- They already think there is a need to change.

- They have previously shown an ability to innovate.
- They are already well respected within the organization.
- They have a broad knowledge of the organization and how it operates.
- Ideally, they already believe that DevOps practices are what is needed.

## Mentoring team members on agile practices

While it's desirable to have formal agile training for staff members, no matter how good any agile course is, there is a world of difference between learning a concept within a few days and putting it into practice.

When they first start an agile transformation, many teams hire external coaches or mentors.

Agile coaches help teams or individuals to adopt agile methods or to improve the current techniques and practices.

They must be agents of change by helping people understand how they work and encouraging them to adopt new approaches.

Agile coaches typically work with more than one team and remove any roadblocks from inside or outside the organization.

This work requires various skills, including coaching, mentoring, teaching, and making easier. Agile coaches must be both trainers and consultants.

There is more than one type of agile coach.

- Some coaches are technical experts who aim to show staff members how to apply specific concepts—for example, test-driven development and continuous integration or deployment.
  - These coaches might do peer programming sessions with staff members.
- Other coaches are focused on agile processes, determining requirements, and managing work activities.
  - They might help how to run effective stand-up and review meetings.
  - Some coaches may themselves act as scrum masters.
    - They might mentor staff in how to fill these roles.

Over time, though, team members need to develop an ability to mentor each other. Teams should aim to be self-organizing. Team members are often expected to learn as they work and to acquire skills from each other. To make it effective, though, the work itself needs to be done collaboratively, not by individuals working by themselves.

## Enable in-team and cross-team collaboration

Completed 100 XP

- 4 minutes

Effective collaboration is critical for well-functioning Agile teams. Enabling it requires cultural changes, cross-functional team collaboration, and tooling.

### Cultural changes

Over recent decades, offices have often become open spaces with few walls. At the time of writing, a significant shift to working from home started as a response to the pandemic. Both situations can limit collaboration, and ambient noise and distractions often reduce productivity. Staff tends to work better when they have comfortable working environments. Defined meeting times and locations let teams choose when they want to interact with others.

Asynchronous communication should be encouraged, but there should not be an expectation that all communications will be responded to urgently. Staff should focus on their primary tasks without feeling like they are being left out of important decisions.

All meetings should have strict timeframes and, more importantly, have a plan. If there is no plan, there should be no meeting.

As it is becoming harder to find the required staff, great teams will be as comfortable with remote or work-from-home as with workers in the office.

To be successful, though, collaboration via communication should become part of the organization's DNA.

Staff should be encouraged to communicate openly and frankly. Learning to deal with conflict is essential for any team, as there will be disagreements at some point. Mediation skills training would be helpful.

## Cross-functional teams

Team members need good collaboration. It is also essential to have a great partnership with wider teams to bring people with different functional expertise together to work toward a common goal.

Often, there will be people from other departments within an organization.

Faster and better innovation can occur in these cross-functional teams.

People from different areas of the organization will have different views of the same problem and are more likely to come up with alternate solutions to problems or challenges. Existing entrenched ideas are more likely to be challenged.

Cross-functional teams can also minimize turf wars within organizations. The more widely a project appears to have ownership, the easier it will be to be widely accepted. Bringing cross-functional teams together also helps to spread knowledge across an organization.

Recognizing and rewarding collective behavior across cross-functional teams can also help to increase team cohesion.

## Collaboration tooling

Agile teams commonly use the following collaboration tools:

[Teams \(Microsoft\)](#): A group chat application from Microsoft. It provides a combined location with workplace chat, meetings, notes, and storage of file attachments. A user can be a member of many teams.

[Slack](#): A commonly used tool for collaboration in Agile and DevOps teams. From a single interface, it provides a series of separate communication channels that can be organized by project, team, or topic. Conversations are kept and are searchable. It is straightforward to add both internal and external team members. Slack integrates with many third-party tools like [GitHub](#) for source code and [DropBox](#) for document and file storage.

[Jira](#): A commonly used tool for planning, tracking, releasing, and reporting.

[Asana](#): A standard tool designed to keep team plans, progress, and discussions in a single place. It has strong capabilities around timelines and boards.

[Clip](#): An offering from Ring Central that provides chat, video, and task management.

Other standard tools with collaboration offerings include ProofHub, RedBooth, Trello, DaPulse, and many others.

## Select tools and processes for agile practices

Completed 100 XP

- 2 minutes

While developing agile methods does not require specific tooling, tools can often enhance the achieved outcomes.

It is essential to realize that the vital tool for agile development is the process itself.

Become familiar with the procedures you need to follow before working out how to implement tools. Several categories of tools are commonly used.

### Physical tools

Not all tools need to be digital tools. Many teams use whiteboards to collaborate on ideas, index cards for recording stories, and sticky notes for moving around tasks.

Even when digital tools are available, it might be more convenient to use these physical tools during stand-up and other meetings.

### Collaboration tools

We described collaboration tools in the previous topic.

### Project management tools

These tools usually include:

- Project planning and execution monitoring abilities (including how to respond to impediments).
- Automation for stand-up meetings.
- Management and tracking of releases.
- A way to record and work with the outcomes of retrospectives.
- Many include Kanban boards and detailed sprint planning options.

These tools will also provide detailed visualizations, often as a graphic dashboard that shows team progress against assigned goals and targets.

Some tools integrate directly with code repositories and CI/CD tools and add code-related metrics, including quality metrics and direct support for code reviews.



As a complete CI/CD system, we have Azure DevOps and GitHub that includes:

- Flexibility in Kanban boards.
- Traceability through Backlogs.
- Customizability in dashboards.
- Built-in scrum boards.
- Integrability directly with code repositories.
- Code changes can be linked directly to tasks or bugs.

Apart from Azure DevOps and GitHub, other standard tools include:

- Jira.
- Trello.
- Active Collab.
- Agilo for Scrum.

- SpiraTeam.
- Icescrum.
- SprintGround.
- Gravity.
- Taiga.
- VersionOne.
- Agilean.
- Wrike.
- Axosoft.
- Assembla.
- PlanBox.
- Asana.
- Binfire.
- Proggio.
- VivifyScrum, and many others.

## **Screen recording tools**

It might seem odd to add screen recording tools to this list. Still, they are beneficial when:

- Work with remote team members.
- Recording bugs in action.
- Building walkthroughs and tutorials that demonstrate actual or potential features.

A screen recorder is built into Windows, but other common ones include SnagIt, Camtasia, OBS, and Loom.

# Choose the DevOps tools

## What is Azure DevOps?

Completed 100 XP

- 2 minutes

Azure DevOps is a Software as a service (SaaS) platform from Microsoft that provides an end-to-end DevOps toolchain for developing and deploying software.

It also integrates with the most-leading tools on the market and is an excellent option for orchestrating a DevOps toolchain.

## What does Azure DevOps provide?

Azure DevOps includes a range of services covering the complete development life cycle.

- Azure Boards: agile planning, work item tracking, visualization, and reporting tool.
- Azure Pipelines: a language, platform, and cloud-agnostic CI/CD platform-supporting containers or Kubernetes.
- Azure Repos: provides cloud-hosted private git repos.
- Azure Artifacts: provides integrated package management with support for Maven, npm, Python, and NuGet package feeds from public or private sources.
- Azure Test Plans: provides an integrated planned and exploratory testing solution.

Also, you can use Azure DevOps to orchestrate third-party tools.

## What if we are not a Microsoft / Microsoft .NET organization?

Azure DevOps is not focused on organizations that are end-to-end Microsoft or Windows.

Azure DevOps provides a platform that is:

- Flexible: you do not have to go 'all in' on Azure DevOps. It is possible to adopt each of the services independently and integrate them with your existing toolchain; most popular tools are supported.
- Cross-Platform: designed to work with any platform (Linux, macOS, and Windows). Or language (including Node.js, Python, Java, PHP, Ruby, C/C++, .NET, Android, and iOS apps). Azure DevOps is not aimed at organizations building and shipping on the Microsoft technology stack.
- Cloud Agnostic: continuous delivery is supported to AWS, GCP, and Azure.

## What is GitHub?

Completed 100 XP

- 1 minute

GitHub is a Software as a service (SaaS) platform from Microsoft that provides Git-based repositories and DevOps tooling for developing and deploying software.

It has a wide range of integrations with other leading tools.

## What does GitHub provide?

GitHub provides a range of services for software development and deployment.

- **Codespaces:** Provides a cloud-hosted development environment (based on Visual Studio Code) that can be operated from within a browser or external tools. Eases cross-platform development.
- **Repos:** Public and private repositories based upon industry-standard Git commands.
- **Actions:** Allows for the creation of automation workflows. These workflows can include environment variables and customized scripts.
- **Packages:** The majority of the world's open-source projects are already contained in GitHub repositories. GitHub makes it easy to integrate with this code and with other third-party offerings.
- **Security:** Provides detailed code scanning and review features, including automated code review assignment.

# Explore an authorization and access strategy

Completed 100 XP

- 3 minutes

Azure DevOps Services uses enterprise-grade authentication. To protect and secure your data, you can use:

- Microsoft account.
- GitHub account.
- Microsoft Entra ID.

Tools like Visual Studio and Azure DevOps natively support the use of Microsoft Accounts and Microsoft Entra ID. Eclipse can also support this form of authentication if you install a Team Explorer Everywhere plug-in.

## Personal access tokens

Use personal access tokens (PAT) for tools that don't directly support Microsoft accounts or Microsoft Entra ID for authentication. You can use it if you want them to integrate with Azure DevOps.

For example, tools like:

- Git-based repositories.
- NuGet.
- Xcode.

These tokens can be set up using Git Credential managers, or you can create them manually.

Personal access tokens are also helpful when establishing access to command-line tools, external tools, and tasks in build pipelines.

Also, when calling REST-based APIs, you don't have a UI popping out to do the authentication. When access is no longer required, you can revoke the personal access token.

## Security groups

Azure DevOps is pre-configured with default security groups.

Default permissions are assigned to the default security groups. You can also configure access at the organization, collection, and project or object levels.

In the organization settings in Azure DevOps, you can configure app access policies. Based on your security policies, you might allow alternate authentication methods, enable third-party applications to access via OAuth, or even allow anonymous access to some projects.

For even tighter control, you can use Conditional Access policies. These offer simple ways to help secure resources such as Azure DevOps when using Microsoft Entra ID for authentication.

## Multifactor authentication

Conditional Access policies such as multifactor authentication can help to minimize the risk of compromised credentials.

As part of a Conditional Access policy, you might require:

- Security group membership.
- A location or network identity.
- A specific operating system.
- A managed device or other criteria.

## Migrate or integrate existing work management tools

Completed 100 XP

- 2 minutes

Both Azure DevOps and GitHub can be integrated with different kinds of work management tools.

As an example, in the Visual Studio Marketplace, Microsoft offers Trello integration tooling.

Migrating from other work management tools to Azure DevOps takes considerable planning.

Most work management tools are highly configurable by the end user. There might not be a tool available that will do the migration without further configuration.

## Jira

Jira is a commonly used work management tool.

In the Visual Studio Marketplace, Solidify offers a tool for Jira to Azure DevOps migration. It migrates in two phases. Jira issues are exported to files, and then the files are imported to Azure DevOps.

If you decide to write the migration code yourself, the following blog post provides a sample code that might help you get started: [Migrate your project from Jira to Azure DevOps](#).

## Other applications

Third-party organizations offer commercial tooling to assist with migrating other work management tools like:

- Aha.
- BugZilla.
- ClearQuest.
- And others to Azure DevOps.

# Migrate or integrate existing test management tools

Completed 100 XP

- 1 minute

Azure Test Plans track manual testing for sprints and milestones, allowing you to follow when that testing is complete.

Azure DevOps also has a Test Feedback extension available in the Visual Studio Marketplace. The extension is used to help teams do exploratory testing and provide feedback.

All team members and other stakeholders can use the extension to submit bugs or provide feedback. For example:

- Developers.
- Product owners.
- Managers.
- UX.
- UI engineers.
- Marketing teams.
- Early adopters.

For load tests, you can use Azure Load Testing. For more information, see [What is Azure Load Testing?](#).

Other helpful testing tools:

Apache JMeter is open-source software written in Java and designed to load test, and measure performance.

Pester is a tool that can automate the testing of PowerShell code.

SoapUI is another testing framework for SOAP and REST testing.

If you are using Microsoft Test Manager, you should plan to migrate to Azure Test Plans instead.

For more information, search for Test Management at Visual Studio Marketplace.

## Design a license management strategy

Completed 100 XP

- 1 minute

When designing a license management strategy, you first need to understand your progress in the DevOps implementation phase.

If you have a draft of the architecture, you're planning for the DevOps implementation; you already know part of the resources to consume.

For example, you started with a version control-implementing Git and created some pipelines to build and release your code.

If you have multiple teams building their solutions, you don't want to wait in the queue to start building yours.

Probably, you want to pay for parallel jobs and make your builds run in parallel without depending on the queue availability.

**To consider:**

- What phase are you in?
- How many people are using the feature?
- How long are you willing to wait if in the queue for pipelines? Is this urgent? Is this a validation only?
- Should all users access all features? Are they Stakeholders? Basic users? Do they already have a Visual Studio license?
- Do you have an advanced Package Management strategy? Maybe you need more space for Artifacts.

For the latest, most up-to-date Azure DevOps pricing information, visit [Azure DevOps Pricing](#).

For the latest, most up-to-date GitHub pricing information, visit [GitHub Pricing](#).

# Plan Agile with GitHub Projects and Azure Boards

## Introduction to GitHub Projects and Project boards

Completed 100 XP

- 3 minutes

### Project Boards

During the application or project lifecycle, it's crucial to plan and prioritize work. With Project boards, you can control specific feature work, roadmaps, release plans, etc.

Project boards are made up of issues, pull requests, and notes categorized as cards you can drag and drop into your chosen columns. The cards contain relevant metadata for issues and pull requests, like labels, assignees, the status, and who opened it.

There are different types of project boards:

- **User-owned project boards:** Can contain issues and pull requests from any personal repository.
- **Organization-wide project boards:** Can contain issues and pull requests from any repository that belongs to an organization.
- **Repository project boards:** Are scoped to issues and pull requests within a single repository.

To create a project board for your organization, you must be an organization member.

It's possible to use templates to set up a new project board that will include columns and cards with tips. The templates can be automated and already configured.

Expand table

### Templates

Basic kanban

### Description

Track your tasks with: To do, In progress, and Done columns.

## Templates

Automated kanban

Automated kanban with review

Bug triage

## Description

Cards automatically move between: To do, In progress, and Done columns.

Cards automatically move between: To do, In progress, and Done columns, with explicit review status.

Triage and prioritize bugs with: To do, High priority, Low priority, and Closed columns.

For more information about Project boards, see:

- [Creating a project board](#).
- [Editing a project board](#).
- [Copying a project board](#).
- [Adding issues and pull requests to a project board](#).
- [Project board permissions for an organization](#).

## Projects

Projects are a new, customizable and flexible tool version of projects for planning and tracking work on GitHub.

A project is a customizable spreadsheet in which you can configure the layout by filtering, sorting, grouping your issues and PRs, and adding custom fields to track metadata.

You can use different views such as Board or spreadsheet/table.

| All work |  | Ready for review | Board            | + New view |              |   |   |
|----------|--|------------------|------------------|------------|--------------|---|---|
| Q        | Title                                      | Assignees        | Status           | Complexity | Target       | ↑ | + |
| 1        | Add Travis CI migration table              | octocat          | Ready for review | 2          | Jun 15, 2021 |   |   |
| 2        | Using a package without installing package | monalisa         | Ready            | 3          | Jun 15, 2021 |   |   |
| 3        | Add functionality to hide images           | octocat          | In Progress      | 2          | Jun 22, 2021 |   |   |
| 4        | Update contributing guide                  | octocat          | Ready for review | 1          | Jun 30, 2021 |   |   |
| 5        | Fix markdown tables in translated content  | monalisa         | Ready            | 1          | Jul 1, 2021  |   |   |
| 6        | add copy button to examples                | octocat          | In Progress      | 1          | Jul 2, 2021  |   |   |
| 7        | Validate Java Gradle wrapper               |                  | Todo             | 2          | Jul 8, 2021  |   |   |
| 8        | GPC creation on linux                      |                  | Todo             | 3          | Jul 18, 2021 |   |   |
| 9        | Add domains for self-hosted runners        |                  | Todo             | 2          | Aug 25, 2021 |   |   |

If you change your pull request or issue, your project reflects that change.

You can use custom fields in your tasks. For example:

- A date field to track target ship dates.
- A number field to track the complexity of a task.
- A single select field to track whether a task is Low, Medium, or High priority.
- A text field to add a quick note.
- An iteration field to plan work week-by-week, including support for breaks.

For more information about Projects, see:

- [Creating a project](#).
- [Managing iterations in projects](#).
- [Customizing your project views](#).
- [Automating projects](#).

## Introduction to Azure Boards

Completed 100 XP

- 3 minutes

Azure Boards is a customizable tool to manage software projects supporting Agile, Scrum, and Kanban processes by default. Track work, issues, and code defects associated with your project. Also, you can create your custom process templates and use them to create a better and more customized experience for your company.

You have multiple features and configurations to support your teams, such as calendar views, configurable dashboards, and integrated reporting.

The Kanban board is one of several tools that allows you to add, update, and filter user stories, bugs, features, and epics.

| Design  | Develop & Test  |   |
|---|---|---|
| Doing<br>1936 Provide related items or frequently bought together section when people browse or search<br>Unassigned<br>Area Path PartsUnlimited<br>Priority 2<br>1                             | Done<br>2055 Decline in orders noticed - Please Investigate immediately<br>Unassigned<br>Severity 3 - Medium<br>Area Path PartsUnlimited                  | Done  |
| Doing<br>1937 As tester, I need to test the website on all the relevant browsers and devices and be sure that it can handle our load.<br>Unassigned 8<br>Area Path PartsUnlimited<br>Priority 2 | Done<br>1950 Notify the user about any changes made to the order<br>Unassigned 10<br>Area Path PartsUnlimited<br>Priority 2<br>1                          | Doing<br>1946 As a customer, I would like to store my credit card details securely<br>Unassigned<br>Area Path PartsUnlimited<br>Priority 2<br>3 |
| Doing<br>1938 As a customer, I should be able to put items to shopping cart.<br>Unassigned 8<br>Area Path PartsUnlimited<br>Priority 2  | Doing<br>1951 As a admin, I should be able to update prices on ad-hoc condition<br>Unassigned 6<br>Area Path PartsUnlimited<br>Priority 2                 | Done<br>1947 As a customer, I should be able to select different shipping option<br>Unassigned 2<br>Area Path PartsUnlimited<br>Priority 1      |
| Doing<br>1939 As a customer, I should be able to print my purchase order<br>Unassigned<br>Area Path PartsUnlimited<br>Priority 2  | Doing<br>1952 As a customer, I would like to provide my feedback on items that I have purchased<br>Unassigned 5<br>Area Path PartsUnlimited<br>Priority 2 | Doing<br>1948 As developer, I want to use Azure Machine Learning to provide a recommendations engine behind the website.<br>Unassigned 2        |

You can track your work using the default work item types such as user stories, bugs, features, and epics. It's possible to customize these types or create your own. Each work item provides a standard set of system fields and controls, including Discussion for adding and tracking comments, History, Links, and Attachments.

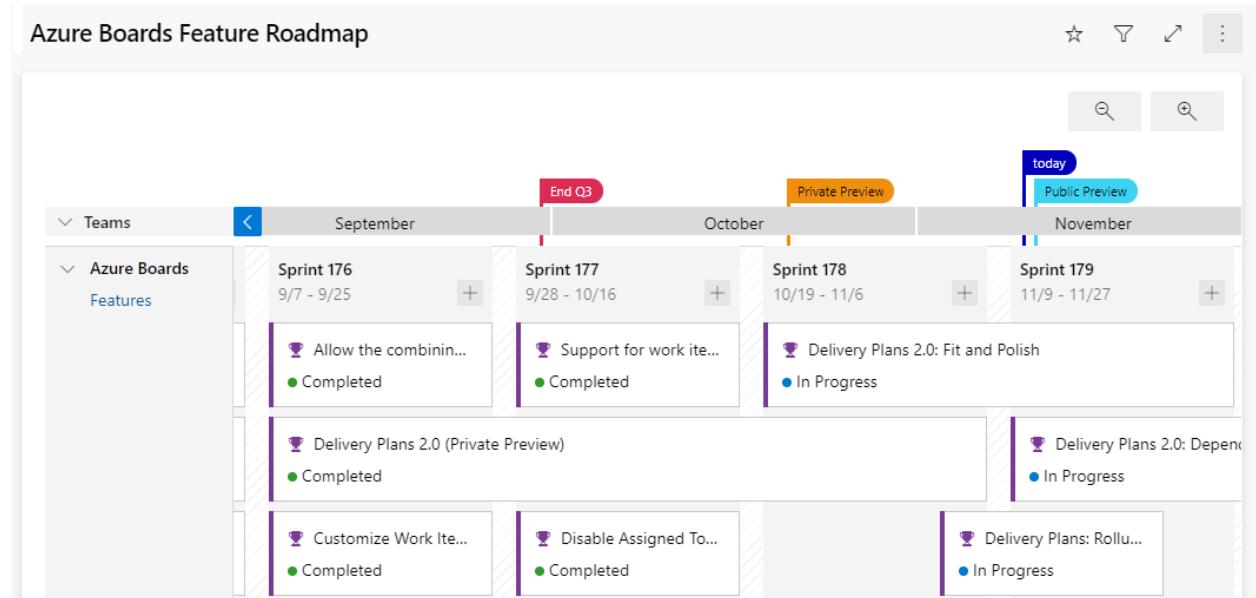
If you need to create reports or a list of work with specific filters, you can use the queries hub to generate custom lists of work items.

Queries support the following tasks:

- Find groups of work items with something in common.
- Triage work to assign to a team member or sprint and set priorities.
- Perform bulk updates.
- View dependencies or relationships between work items.
- Create status and trend charts that you can optionally add to dashboards.

## Delivery plans

It's possible to create another view with deliverables and track dependencies across several teams in a calendar view using Delivery Plans.



Delivery plans are fully interactive, supporting the following tasks:

- View up to 15 team backlogs, including a mix of backlogs and teams from different projects.
- View custom portfolio backlogs and epics.
- View work that spans several iterations.
- Add backlog items from a plan.
- View rollup progress of features, epics, and other portfolio items.
- View dependencies that exist between work items.

For more information about Azure Boards, see:

- [Azure Boards documentation | Microsoft Docs](#).
- [Reasons to start using Azure Boards | Microsoft Docs](#).
- [GitHub and Azure Boards](#).

## Link GitHub to Azure Boards

Completed 100 XP

- 2 minutes

## Use GitHub, track work in Azure Boards

Use Azure Boards to plan and track your work and GitHub as source control for software development.

Connect Azure Boards with GitHub repositories, enabling linking GitHub commits, pull requests, and issues to work items in Boards.

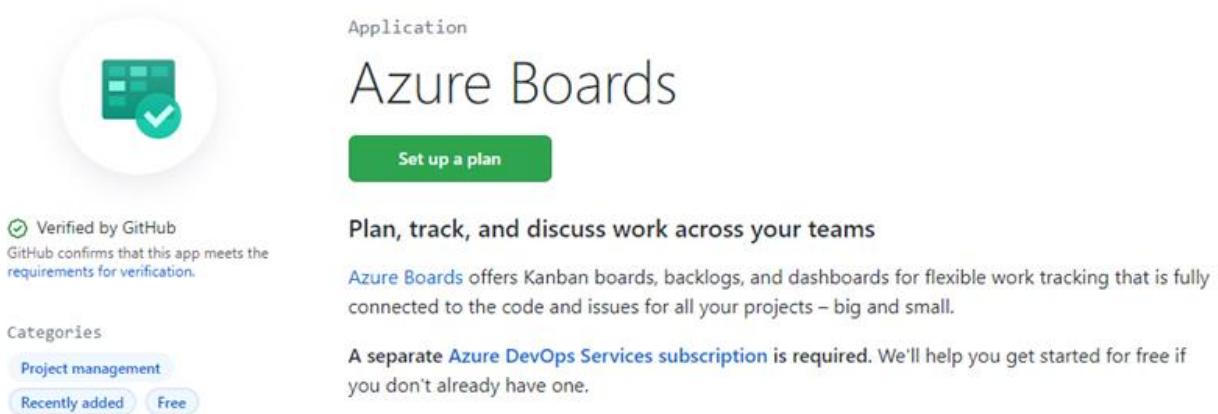
The screenshot shows a Kanban board in Azure Boards. The columns are To Do, Doing, and Done. The Doing column has 1/5 work items. The To Do column has two items: 118 Publish the blog post announcing the new app (State: To Do) and 116 Update our marketing website about the new app (State: To Do). The Done column has one item: 117 Release the app in private preview with early adopters (State: Done). Each work item card links to its GitHub commit details.

## Azure Boards App

The integration is created using the Azure Boards App, acting as a bridge between Azure Boards and GitHub.

To install the app, you must be an administrator or owner of the GitHub repository or the GitHub organization.

The app is installed from the GitHub Marketplace. [Azure Boards App](#)



The screenshot shows the Azure Boards application page in the Microsoft Marketplace. At the top left is a circular icon containing a green checkmark and a small board icon. To its right, the word "Application" is written above the app name "Azure Boards". Below the name is a green button labeled "Set up a plan". On the left side, there's a section titled "Categories" with "Project management" listed, and two status indicators: "Recently added" and "Free". To the right of the categories, there's a "Verified by GitHub" badge with a checkmark, followed by the text: "GitHub confirms that this app meets the requirements for verification." Below this, under the heading "Plan, track, and discuss work across your teams", it says: "Azure Boards offers Kanban boards, backlogs, and dashboards for flexible work tracking that is fully connected to the code and issues for all your projects – big and small." At the bottom of this section, it notes: "A separate [Azure DevOps Services subscription](#) is required. We'll help you get started for free if you don't already have one."

## Authenticating to GitHub

Azure Boards can connect to GitHub. For GitHub in the cloud, when adding a GitHub connection, the authentication options are:

- Username/Password
- Personal Access Token (PAT)

For a walkthrough on making the connection, see: [Connect Azure Boards to GitHub](#).

You can configure other Azure Boards/Azure DevOps Projects, GitHub.com repositories, or change the current configuration from the Azure Boards app page.

Once you've integrated Azure Boards with GitHub using the Azure Boards app, you can add or remove repositories from the web portal for Azure Boards.

## Supported integration scenarios

Azure Boards-GitHub integration supports the following connections:

- From GitHub:
  - Support integration for all repositories for a GitHub account or organization or select repositories.
  - Add or remove GitHub repositories participating in the integration and configure the project they connect to.
  - Suspend Azure Boards-GitHub integration or uninstall the app.

- From Azure Boards:
  - Connect one or more GitHub repositories to an Azure Boards project.
  - Add or remove GitHub repositories from a GitHub connection within an Azure Boards project.
  - Completely remove a GitHub connection for a project.
  - Allow a GitHub repository to connect to one or more Azure Boards projects within the same Azure DevOps organization or collection.

Azure Boards-GitHub integration supports the following operational tasks:

- Create links between work items and GitHub commits, pull requests, and issues based on GitHub mentions.
- Support state transition of work items to a Done or Completed state when using GitHub mention by using fix, fixes, or fixed.
- Support full traceability by posting a discussion comment to GitHub when linking from a work item to a GitHub commit, pull request, or issue.
- Show linked to GitHub code artifacts within the work item Development section.
- Show linked to GitHub artifacts as annotations on Kanban board cards.
- Support status badges of Kanban board columns added to GitHub repositories.

The following tasks aren't supported at this time:

- Query for work items with links to GitHub artifacts. However, you can query for work items with an External Link Count greater than 0.

## **Note**

Reference: [Azure Boards-GitHub integration](#).

For more information, see:

- [Change GitHub repository access, or suspend or uninstall the integration](#).
- [Add or remove GitHub repositories](#).
- [Link GitHub commits, pull requests, and issues to work items for details on linking to work items](#).

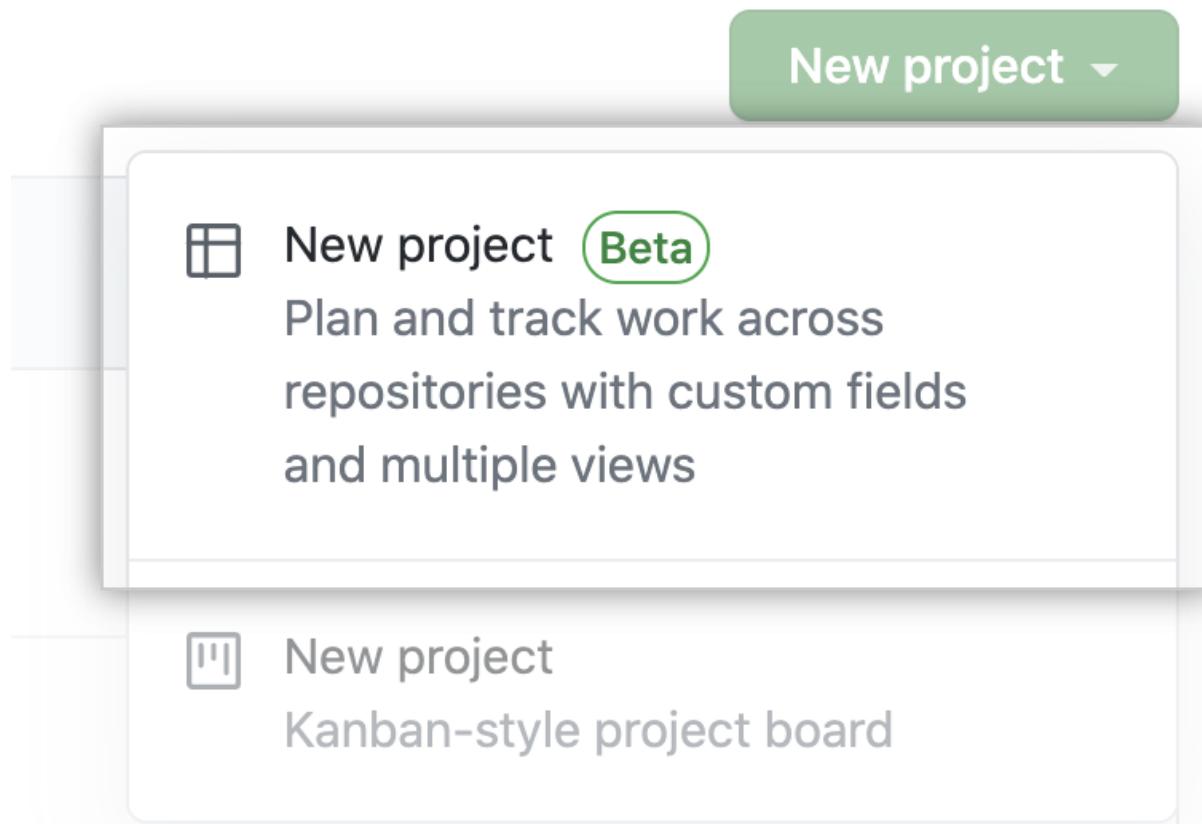
# Configure GitHub Projects

Completed 100 XP

- 3 minutes

## Create a project

To start working with GitHub Projects, you first need to create an organization or user project.



To create an **organization project**:

1. On GitHub, navigate to the main page of your organization.
2. Click **Projects**.
3. Select the **New project** drop-down menu and click **New project**.

To create a **user project**:

1. On any GitHub page, click on your avatar, then select **Your projects**.
2. Select the **New project** drop-down menu and click **New project**.

Create a project **description** or a **README** file:

1. Navigate to your project.
2. In the top-right, click to open the menu.
3. In the menu, click **Settings**.
4. Under Add a description, type your description in the text box and click **Save**.
5. To update your project's README, type your content in the text box under README.
6. Click Save.

## Adding issues

When your new project initializes, it prompts you to add items.

Click on the plus (+) sign to add more issues.

The screenshot shows a GitHub project page titled "DevOps Journey". At the top, there is a search bar and a menu with options like Pulls, Issues, Marketplace, and Explore. On the right, there are notifications, a "+" button, and user profile icons. Below the title, there are three view options: "View 1", "View 3", and "+ New view". A "Beta" button and a "Give feedback" link are also present. The main area displays a single issue titled "Adding new issues". Below this, there is a draft issue input field containing the placeholder "Type to add a draft issue or use # to search". A command palette dropdown is open, listing "Add an issue from a repository", "Command palette", and "Help and documentation". Shortcuts for "ctrl + k" are shown to the right of the palette.

## Project settings and permissions

You can view and make quick changes to your project description and README by navigating to your project and clicking on the top right.

The screenshot shows the GitHub Project Settings page for a project named "DevOps Journey".

- Project settings:** A section containing "Manage access" and "Custom fields". The "Custom fields" section is highlighted with a red box and contains "Status" and "Target Date" options. A red arrow points from the "Danger zone" section up towards this area.
- Add a description:** A section where you can "Add a short description". It is highlighted with a red box.
- README:** A section where you can preview and edit the README content. It is highlighted with a red box. The text "Markdown is supported" is visible at the bottom.
- Danger zone:** A section containing "Visibility" (set to Private), "Close project" (with a "Close this project" button), and other project management options.

You can create custom fields to add to your project.

## ← Settings

Project settings

Manage access

Custom fields

Status

Target Date

+ New field

## Project settings

Project name

DevOps Journey

Release Date

Date

Aa Text

1/3 Number

Date

Single select

Iteration

Save

Preview

everyone know what t

Manage access and permissions by adding collaborators.

## Who has access

### Private project



Only those with access to this project can view it.

[Manage](#)

## Invite collaborators

Search by username

Role: Write ▾

[Invite](#)

## Manage access

0 members

Type ▾

Role ▾

[Find a collaborator](#)

You don't have any collaborators yet.

Add a collaborator to see them here.

For more information about Projects, see:

- [Quickstart for projects - GitHub Docs](#).
- [Creating an issue - GitHub Docs](#).

# Manage work with GitHub Project boards

Completed 100 XP

- 2 minutes

GitHub Projects allow you to control project deliverables, release dates, and iterations to plan upcoming work.

You can create an iteration to:

- Associate items with specific repeating blocks of time.
- Set to any length of time.
- Include breaks.

It's possible to configure your project to group by iteration to visualize the balance of upcoming work.

When you first create an iteration field, three iterations are automatically created. You can add other iterations if needed.

The screenshot shows the GitHub Project settings interface. On the left sidebar, under 'Custom fields', 'New Field 2' is selected. The main area displays 'New Field 2 field settings' with three iterations listed: 'New Field 2 1' (Planned, 2 weeks, May 30 - Jun 12), 'New Field 2 2' (Planned, 2 weeks, Jun 13 - Jun 26), and 'New Field 2 3' (Planned, 2 weeks, Jun 27 - Jul 10). At the bottom, there are 'Save changes' and 'Reset' buttons.

| Iteration     | Status  | Duration                |
|---------------|---------|-------------------------|
| New Field 2 1 | Planned | 2 weeks May 30 - Jun 12 |
| New Field 2 2 | Planned | 2 weeks Jun 13 - Jun 26 |
| New Field 2 3 | Planned | 2 weeks Jun 27 - Jul 10 |

## Iteration field

You can use the command palette or the project's interface to create an iteration field.

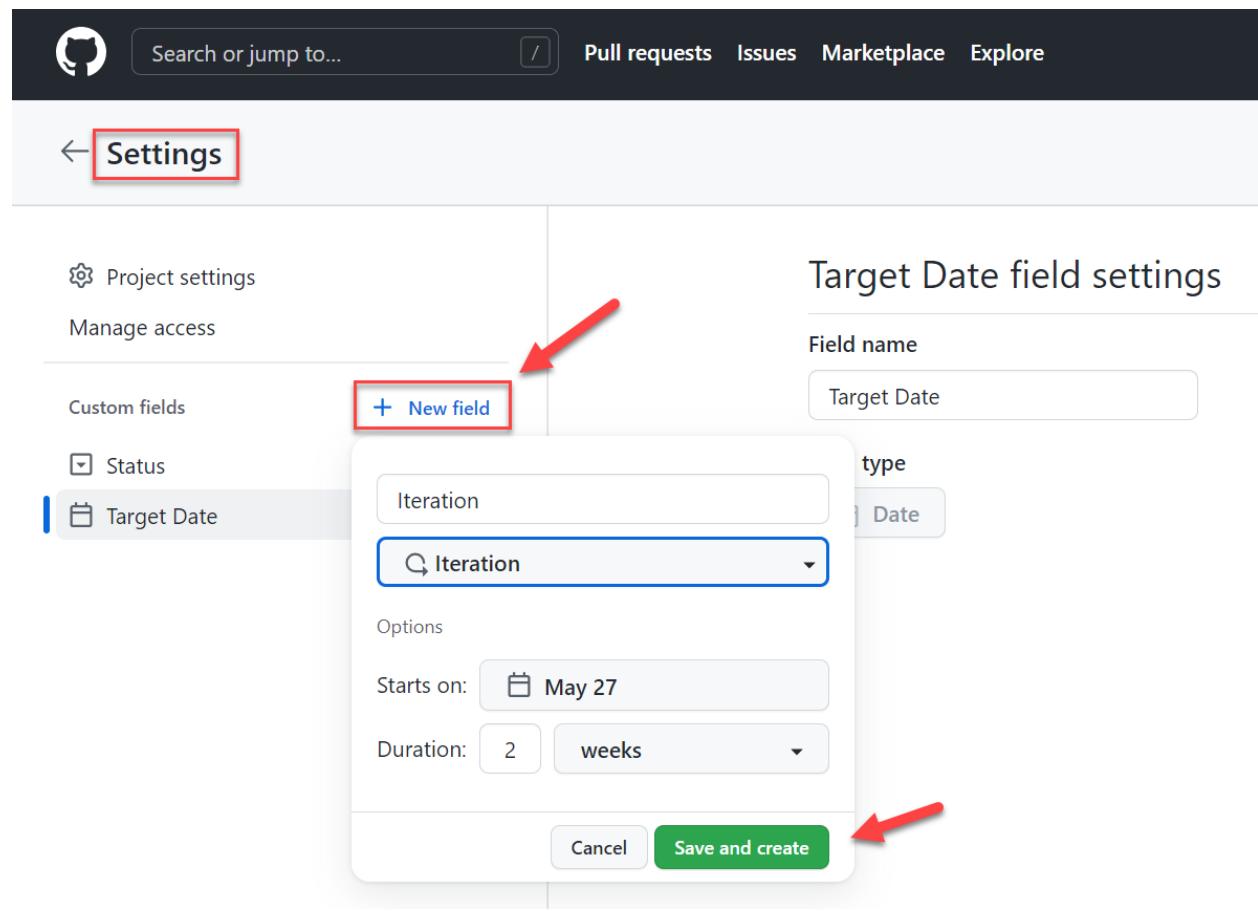
### Tip

To open the project command palette, press Ctrl+K (Windows/Linux) or Command+K (Mac).

Start typing any part of "Create new field". When "Create new field" displays in the command palette, select it.

Or follow the steps using the interface:

1. Navigate to your project.
2. Click in the plus (+) sign in the rightmost field header. A drop-down menu with the project fields will appear.
3. Click in the **New field**.
4. Enter a name for the new iteration field.
5. Select the dropdown menu below and click Iteration.
6. (Optional) Change the starting date from the current day, select the calendar dropdown next to Starts on, and click on a new starting date.
7. To change the duration of each iteration, type a new number, then select the dropdown and click either days or weeks.
8. Click Save and create.



## Adding new iterations

1. Navigate to your project.
2. In the top-right, click to open the menu.
3. In the menu, click **Settings** to access the project settings.
4. Click the name of the iteration field you want to adjust.
5. To add a new iteration of the same duration, click **Add iteration**.
6. (Optional) Customize the duration of the new iteration and when it starts.
  - a. Click next to Add iteration.
  - b. Select a starting date and duration.
  - c. Click **Add**.
7. Click Save changes.

The screenshot shows the 'Iterations' section of a GitHub Project settings page. At the top, there are counts for '3 Active' and '4 Completed' iterations. A button for '+ Add iteration' is in the top right. Below this, four iterations are listed:

- Iteration 5** (Current): 2 weeks, Feb 23 - Mar 08. An 'Insert break' button is to its right.
- Iteration 6** (Planned): 2 weeks, Mar 09 - Mar 22. A trash icon is to its right.
- Break**: 1 week, Mar 23 - Mar 29. A trash icon is to its right.
- Iteration 7** (Planned): 2 weeks, Mar 30 - Apr 12. A trash icon is to its right.

At the bottom are 'Save changes' and 'Reset' buttons.

Also, you can insert breaks into your iterations to communicate when you're taking time away from scheduled work.

For more information about iterations, see:

- [Managing iterations in projects \(beta\) - GitHub Docs](#).
- [Best practices for managing projects \(beta\) - GitHub Docs](#).

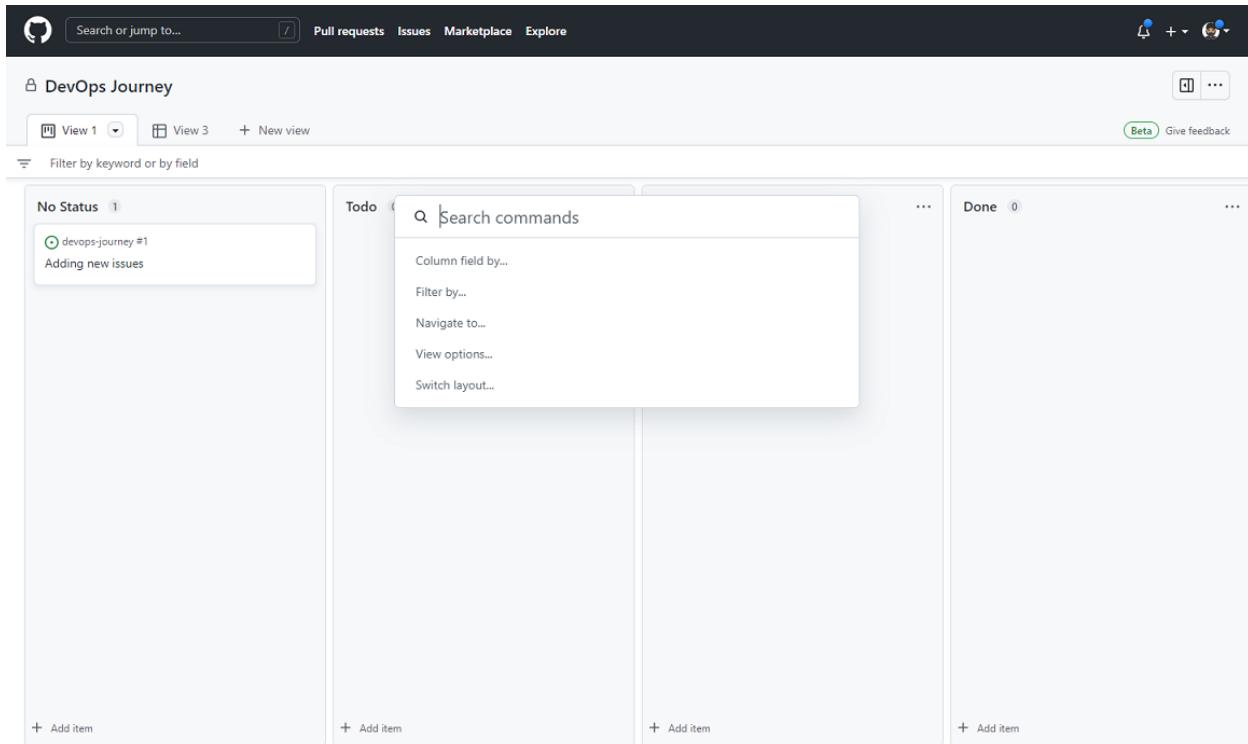
## Customize Project views

Completed 100 XP

- 3 minutes

Using Projects views, you can organize information by changing the layout, grouping, sorting, and filtering your work.

You can create and use different visualizations, for example, Board view:



## Project command palette

Use the project command palette to change settings and run commands in your project.

1. Open the project command palette by pressing **Command + K (Mac)** or **Ctrl + K (Windows/Linux)**.
2. Type any command part or navigate through the command palette window to find a command.

You have multiple commands to apply, such as:

- Switch layout: Table.
- Show: Milestone.
- Sort by: Assignees, asc.
- Remove sort-by.
- Group by: Status.
- Remove group-by.
- Column field by: Status.
- Filter by Status.
- Delete view.

### Note

For more information about GitHub Command Palette, see [GitHub Command Palette - GitHub Docs](#).

Also, you can perform changes using the interface.

## Creating a project view

Project views allow you to view specific aspects of your project. Each view is displayed on a separate tab in your project.

For example, you can have:

- A view that shows all items not yet started (filter on Status).
- A view that shows the workload for each team (group by a custom Team field).
- A view that shows the items with the earliest target ship date (sort by a date field).

To add a new view:

1. To open the project command palette, press **Command + K (Mac)** or **Ctrl + K (Windows/Linux)**.
2. Start typing **New view** (to create a new view) or **Duplicate view** (to duplicate the current view).
3. Choose the required command.
4. The new view is automatically saved.

For more information about projects (beta), see:

- [About projects \(beta\)](#).
- [Creating a project \(beta\)](#).

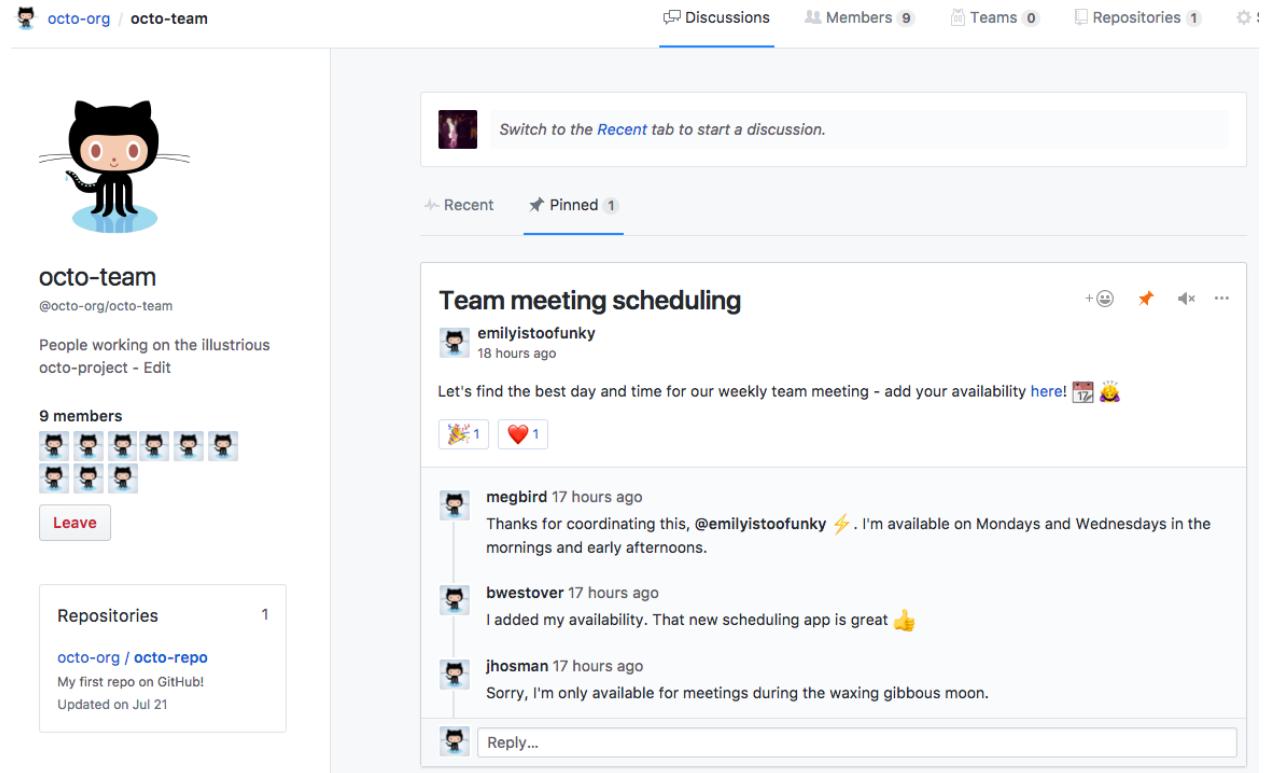
## Collaborate using team discussions

Completed 100 XP

- 3 minutes

GitHub discussions can help make your team plan together, update one another, or talk about any topic you'd like in discussion posts on your team's page in an organization.

You can use team discussions for conversations that span across projects or repositories (issues, pull requests, etc.). Instead of opening an issue in a repository to discuss an idea, you can include the entire team by having a conversation in a team discussion.

A screenshot of the GitHub web interface showing a team discussion. On the left, there's a sidebar for the 'octo-team' in the 'octo-org' organization. It shows a team icon (a cat), the team name, a description ('People working on the illustrious octo-project'), and a member count of 9. Below that is a 'Leave' button and a 'Repositories' section with one item. On the right, the main area is titled 'Discussions'. A message says 'Switch to the Recent tab to start a discussion.' Below it, the 'Recent' tab is selected, showing a pinned post from 'emilyistoofunky' about scheduling a team meeting. Other team members like 'megbird', 'bwestover', and 'jhosman' have responded. There are also icons for adding a file, a star, a comment, and more options.

With team discussions, you can:

- Post on your team's page or participate in a public discussion.
- Link to any team discussion to reference it elsewhere.
- Pin important posts to your team's page.
- Receive email or web notifications.

Team discussions are available in organizations by default.

You can also use organization discussions to facilitate conversations across your organization.

For more information about team discussion, see:

- [Enabling or disabling GitHub Discussions for an organization](#).
- [Quickstart for communicating on GitHub](#).
- [About teams](#).
- [Creating a team discussion](#).

- [Editing or deleting a team discussion.](#)

# Agile Plan and Portfolio Management with Azure Boards

Completed 100 XP

- 57 minutes

**Estimated time:** 60 minutes.

**Lab files:** none.

## Scenario

In this lab, you'll learn about the agile planning and portfolio management tools and processes provided by Azure Boards and how they can help you quickly plan, manage, and track work across your entire team. You'll explore the product backlog, sprint backlog, and task boards that can track the flow of work during an iteration. We'll also look at the enhanced tools in this release to scale for larger teams and organizations.

## Objectives

After completing this lab, you'll be able to:

- Manage teams, areas, and iterations.
- Manage work items.
- Manage sprints and capacity.
- Customize Kanban boards.
- Define dashboards.
- Customize team process.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).

## **Exercises**

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Manage an Agile project.
- Exercise 2 (optional): Define dashboards.

# Introduction to source control

## Explore DevOps foundational practices

Completed 100 XP

- 3 minutes

The [State of DevOps Report 2021](#) highlights version control in almost all stages of DevOps evolution.

### Foundational practices and the 5 stages of DevOps evolution

|         | Defining practices* and associated practices  | Practices that contribute to success   |
|---------|---|--|
| Stage 0 | <ul style="list-style-type: none"><li>Monitoring and alerting are configurable by the team operating the service.</li><li>Deployment patterns for building applications or services are reused.</li><li>Testing patterns for building applications or services are reused.</li><li>Teams contribute improvements to tooling provided by other teams.</li><li>Configurations are managed by a configuration management tool.</li></ul> |  |
| Stage 1 | <ul style="list-style-type: none"><li><b>Application development teams use version control.</b></li><li>Teams deploy on a standard set of operating systems.</li></ul>  | <ul style="list-style-type: none"><li>Build on a standard set of technology.</li><li>Put application configurations in version control.</li><li>Test infrastructure changes before deploying to production.</li><li>Source code is available to other teams.</li></ul> |
| Stage 2 | <ul style="list-style-type: none"><li>Build on a standard set of technology.</li><li>Teams deploy on a single standard operating system.</li></ul>  | <ul style="list-style-type: none"><li>Deployment patterns for building applications and services are reused.</li><li>Rearchitect applications based on business needs.</li><li>Put system configurations in version control.</li></ul>                                 |

It is helpful for non-developers in an organization to understand the fundamentals of the discipline.

Also, it is so deeply rooted in the daily life of software engineers. Essential if those individuals decide which version control tools and platforms to use.

|         |   |  |
|---------|---|--|
| Stage 3 | <ul style="list-style-type: none"><li>Individuals can do work without manual approval from outside the team.</li><li>Deployment patterns for building applications and services are reused.</li><li>Infrastructure changes are tested before deploying to production.</li></ul> | <ul style="list-style-type: none"><li>Individuals can make changes without significant wait times.</li><li>Service changes can be made during business hours.</li><li>Post-incident reviews occur and results are shared.</li><li>Teams build on a standard set of technologies.</li><li>Teams use continuous integration.</li><li>Infrastructure teams use version control.</li></ul> |
| Stage 4 | <ul style="list-style-type: none"><li>System configurations are automated.</li><li>Provisioning is automated.</li><li>Application configurations are in version control.</li><li>Infrastructure teams use version control.</li></ul>  | <ul style="list-style-type: none"><li>Security policy configurations are automated.</li><li>Resources made available via self-service.</li></ul>   |
| Stage 5 | <ul style="list-style-type: none"><li>Incident responses are automated.</li><li>Resources available via self-service.</li><li>Rearchitect applications based on business needs.</li><li>Security teams are involved in technology design and deployment.</li></ul>              | <ul style="list-style-type: none"><li>Security policy configurations are automated.</li><li>Application developers deploy testing environments on their own.</li><li>Success metrics for projects are visible.</li><li>Provisioning is automated.</li></ul>  |

\* The practices that define each stage are highlighted in bold font.

Version control is essential for all software development projects and is vital at large businesses and enterprises.

Enterprises have many stakeholders. For example:

- Distributed teams.
- Strict processes and workflows.
- Siloed organizations.
- Hierarchical organizations.

All those characteristics represent coordination and integration challenges when it comes to merging and deploying code.

Companies within highly regulated industries need a practical way to ensure that all standards are met appropriately and mitigate risk—for example, banking and healthcare.

## What is source control?

Completed 100 XP

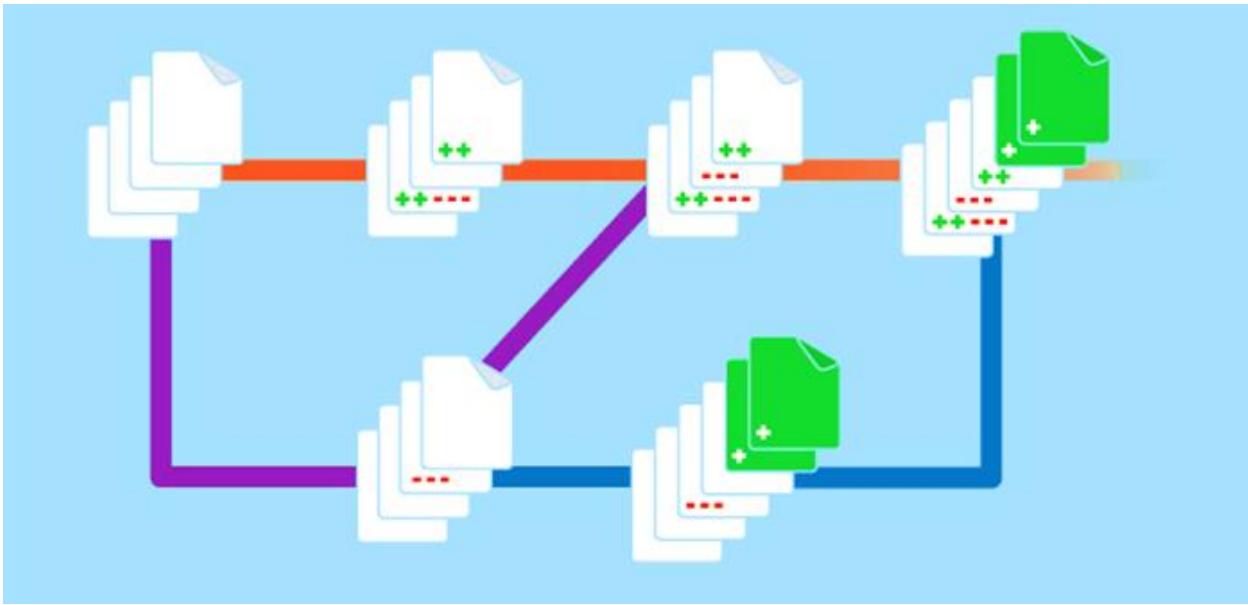
- 2 minutes

A Source control system (or version control system) allows developers to collaborate on code and track changes. Use version control to save your work and coordinate code changes across your team. Source control is an essential tool for multi-developer projects.

The version control system saves a snapshot of your files (history) so that you can review and even roll back to any version of your code with ease. Also, it helps to resolve conflicts when merging contributions from multiple sources.

For most software teams, the source code is a repository of invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort.

Source control protects source code from catastrophe and the casual degradation of human error and unintended consequences.



Without version control, you're tempted to keep multiple copies of code on your computer. It could be dangerous. Easy to change or delete a file in the wrong code copy, potentially losing work.

Version control systems solve this problem by managing all versions of your code but presenting you with a single version at a time.

Tools and processes alone aren't enough to accomplish the above, such as adopting Agile, Continuous Integration, and DevOps. Believe it or not, all rely on a solid version control practice.

Version control is about keeping track of every change to software assets—tracking and managing the who, what, and when. Version control is the first step needed to assure quality at the source, ensure flow and pull value, and focus on the process. All of these create value not just for the software teams but ultimately for the customer.

Version control is a solution for managing and saving changes made to any manually created assets. If changes are made to the source code, you can go back in time and easily roll back to previous-working versions.

Version control tools will enable you to see who made changes, when, and what exactly was changed.

Version control also makes experimenting easy and, most importantly, makes collaboration possible. Without version control, collaborating over source code would be a painful operation.

There are several perspectives on version control.

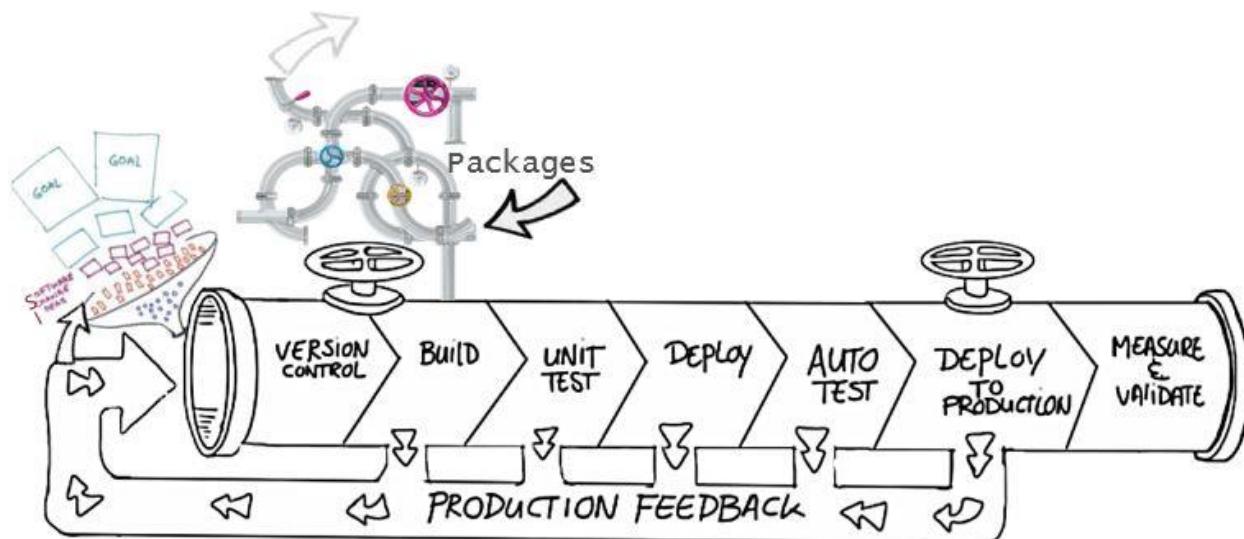
- For developers, it's a daily enabler for work and collaboration to happen. It's part of the daily job, one of the most-used tools.
- For management, the critical value of version control is in:
  - IP security.
  - Risk management.
  - Time-to-market speed through Continuous Delivery, where version control is a fundamental enabler.

## Explore benefits of source control

Completed 100 XP

- 3 minutes

"Code does not exist unless it is committed into source control. Source control is the fundamental enabler of continuous delivery."



Whether writing code professionally or personally, you should always version your code using a source control management system. Some of the advantages of using source control are,

- **Create workflows.** Version control workflows prevent the chaos of everyone using their development process with different and incompatible tools. Version control systems provide process enforcement and permissions, so everyone stays on the same page.

- **Work with versions.** Every version has a description in the form of a comment. These descriptions help you follow changes in your code by version instead of by individual file changes. Code stored in versions can be viewed and restored from version control at any time as needed. It makes it easy to base new work on any version of code.
- **Collaboration.** Version control synchronizes versions and makes sure that your changes do not conflict with other changes from your team. Your team relies on version control to help resolve and prevent conflicts, even when people make changes simultaneously.
- **Maintains history of changes.** Version control keeps a record of changes as your team saves new versions of your code. This history can be reviewed to find out who, why, and when changes were made. The history gives you the confidence to experiment since you can roll back to a previous good version at any time. The history lets your base work from any code version, such as fixing a bug in an earlier release.
- **Automate tasks.** Version control automation features save your team time and generate consistent results. Automate testing, code analysis, and deployment when new versions are saved to version control.

## Common software development values

- **Reusability** – why do the same thing twice? Reuse of code is a common practice and makes building on existing assets simpler.
- **Traceability** – Audits are not just for fun; in many industries, it is a legal matter. All activities must be traced, and managers can produce reports when needed. Traceability also makes debugging and identifying root cause easier. Additionally, it helps with feature reuse as developers can link requirements to implementation.
- **Manageability** – Can team leaders define and enforce workflows, review rules, create quality gates and enforce QA throughout the lifecycle?
- **Efficiency** – are we using the right resources for the job, minimizing time and effort? This one is self-explanatory.
- **Collaboration** – When teams work together, quality tends to improve. We catch one another's mistakes and can build on each other's strengths.
- **Learning** – Organizations benefit when they invest in employees learning and growing. It is important for onboarding new team members, the lifelong learning of seasoned members, and the opportunity for workers to contribute to the bottom line and the industry.

# Explore best practices for source control

Completed 100 XP

- 2 minutes
  - **Make small changes.** In other words, commit early and commit often. Be careful not to commit any unfinished work that could break the build.
  - **Do not commit personal files.** It could include application settings or SSH keys. Often personal files are committed accidentally but cause problems later when other team members work on the same code.
  - **Update often and right before pushing to avoid merge conflicts.**
  - **Verify your code change before pushing it to a repository;** ensure it compiles and tests are passing.
  - **Pay close attention to commit messages, as it will tell you why a change was made.** Consider committing messages as a mini form of documentation for the change.
  - **Link code changes to work items.** It will concretely link what was created to why it was created—or modified by providing traceability across requirements and code changes.
  - **No matter your background or preferences, be a team player and follow agreed conventions and workflows.** Consistency is essential and helps ensure quality, making it easier for team members to pick up where you left off, review your code, debug, and so on.

Using version control of some kind is necessary for any organization, and following the guidelines can help developers avoid needless time spent fixing errors and mistakes.

These practices also help organizations reap more significant benefits from having a good version control system.

# Describe types of source control systems

## Understand centralized source control

Completed 100 XP

- 2 minutes



| Strengths                              | Best used for                             |
|--|---|
| Easily scales for very large codebases | Large integrated codebases                |
| Granular permission control            | Audit & Access control down to file level |
| Permits monitoring of usage            | Hard to merge file types                  |
| Allows exclusive file locking          |   |

Centralized source control systems are based on the idea that there's a single "central" copy of your project somewhere (probably on a server). Programmers will check in (or commit) their changes to this central copy.

"Committing" a change means to record the difference in the central system. Other programmers can then see this change.

Also, it's possible to pull down the change. The version control tool will automatically update the contents of any files that were changed.

Most modern version control systems deal with "changesets," which are a group of changes (possibly too many files) that should be treated as a cohesive whole.

Programmers no longer must keep many copies of files on their hard drives manually. The version control tool can talk to the central copy and retrieve any version they need on the fly.

Some of the most common-centralized version control systems you may have heard of or used are Team Foundation Version Control (TFVC), CVS, Subversion (or SVN), and Perforce.

## A typical centralized source control workflow

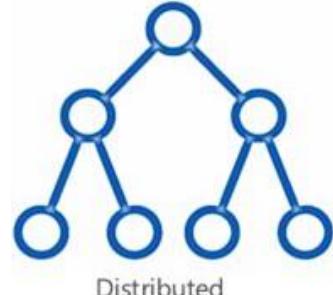
If working with a centralized source control system, your workflow for adding a new feature or fixing a bug in your project will usually look something like this:

- Get the latest changes other people have made from the central server.
- Make your changes, and make sure they work correctly.
- Check in your changes to the main server so that other programmers can see them.

## Understand distributed source control

Completed 100 XP

- 3 minutes



| Strengths   | Best used for                  |
|---|--------------------------------|
| Cross Platform support                                      | Small & Modular codebases      |
| An open source friendly code review model via pull requests | Evolving through open source   |
| Complete offline support                                    | Highly distributed teams       |
| Portable history  | Teams working across platforms |
| An enthusiastic growing user base                           | Green field codebases          |

Over time, so-called "distributed" source control or version control systems (DVCS for short) have become the most important.

The three most popular are Git, Mercurial, and Bazaar. These systems don't necessarily rely on a central server to store all the versions of a project's files. Instead, every developer "clones" a repository copy and has the project's complete history on their local storage. This copy (or "clone") has all the original metadata.

This method may sound wasteful but it isn't a problem in practice. Most programming projects consist primarily of plain text files (maybe a few images).

The disk space is so cheap that storing many copies of a file doesn't create a noticeable dent in a local storage free space. Modern systems also compress the files to use even less space; for example, objects (and deltas) are stored compressed, and text files used in programming compress well (around 60% of original size, or 40% reduction in size from compression).

Getting new changes from a repository is called "pulling." Moving your changes to a repository is called "pushing." You move changesets (changes to file groups as coherent wholes), not single-file diffs.

One common misconception about distributed version control systems is that there can't be a central project repository. It isn't true. Nothing stops you from saying, "this copy of the project is the authoritative one."

It means that instead of a central repository required by your tools, it's now optional.

## Advantages over centralized source control

The act of cloning an entire repository gives distributed source control tools several advantages over centralized systems:

- Doing actions other than pushing and pulling changesets is fast because the tool only needs to access the local storage, not a remote server.
- Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
- Everything but pushing and pulling can be done without an internet connection. So, you can work on a plane, and you won't be forced to commit several bug fixes as one large changeset.
- Since each programmer has a full copy of the project repository, they can share changes with one, or two other people to get feedback before showing the changes to everyone.

## Disadvantages compared to centralized source control

There are almost no disadvantages to using a distributed source control system over a centralized one.

Distributed systems don't prevent you from having a single "central" repository; they provide more options.

There are only two major inherent disadvantages to using a distributed system:

- If your project contains many large, binary files that can't be efficiently compressed, the space needed to store all versions of these files can accumulate quickly.
- If your project has a long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

# Explore Git and Team Foundation Version Control

Completed 100 XP

- 2 minutes

## Git (distributed)

Git is a distributed version control system. Each developer has a copy of the source repository on their development system. Developers can commit each set of changes on their dev machine.

Branches are lightweight. When you need to switch contexts, you can create a private local branch. You can quickly switch from one branch to another to pivot among different variations of your codebase. Later, you can merge, publish, or dispose of the branch.

## Team Foundation Version Control (TFVC-centralized)

Team Foundation Version Control (TFVC) is a centralized version control system.

Typically, team members have only one version of each file on their dev machines. Historical data is maintained only on the server. Branches are path-based and created on the server.

TFVC has two workflow models:

- **Server workspaces** - Before making changes, team members publicly check out files. Most operations require developers to be connected to the server. This system helps lock workflows. Other software that works this way includes Visual Source Safe, Perforce, and CVS. You can scale up to huge

codebases with millions of files per branch—also, large binary files with server workspaces.

- **Local workspaces** - Each team member copies the latest codebase version with them and works offline as needed. Developers check in their changes and resolve conflicts as necessary. Another system that works this way is Subversion.

## Examine and choose Git

Completed 100 XP

- 4 minutes

Switching from a centralized version control system to Git changes the way your development team creates software.

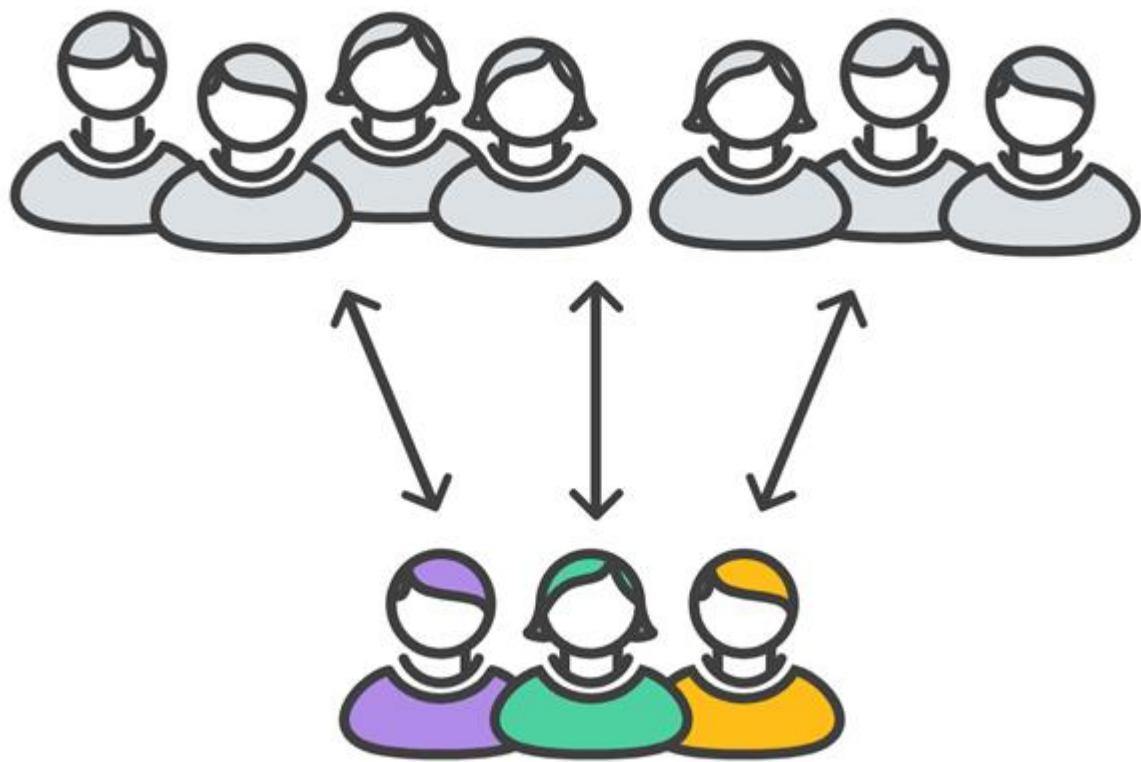
If you are a company that relies on its software for mission-critical applications, altering your development workflow impacts your entire business.

Developers would gain the following benefits by moving to Git.

## Community

In many circles, Git has come to be the expected version control system for new projects.

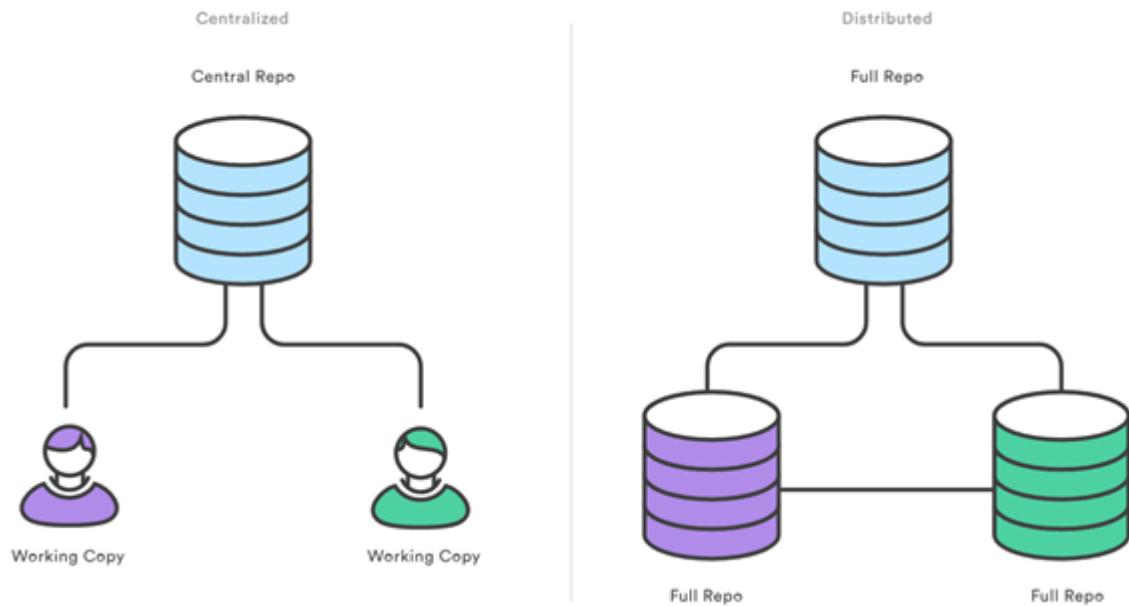
If your team is using Git, odds are you will not have to train new hires on your workflow because they will already be familiar with distributed development.



Also, Git is popular among open-source projects. It is easy to use 3rd-party libraries and encourage others to fork your open-source code.

## Distributed development

In TFVC, each developer gets a working copy that points back to a single central repository. Git, however, is a distributed version control system. Instead of a working copy, each developer gets their local repository, complete with an entire history of commits.



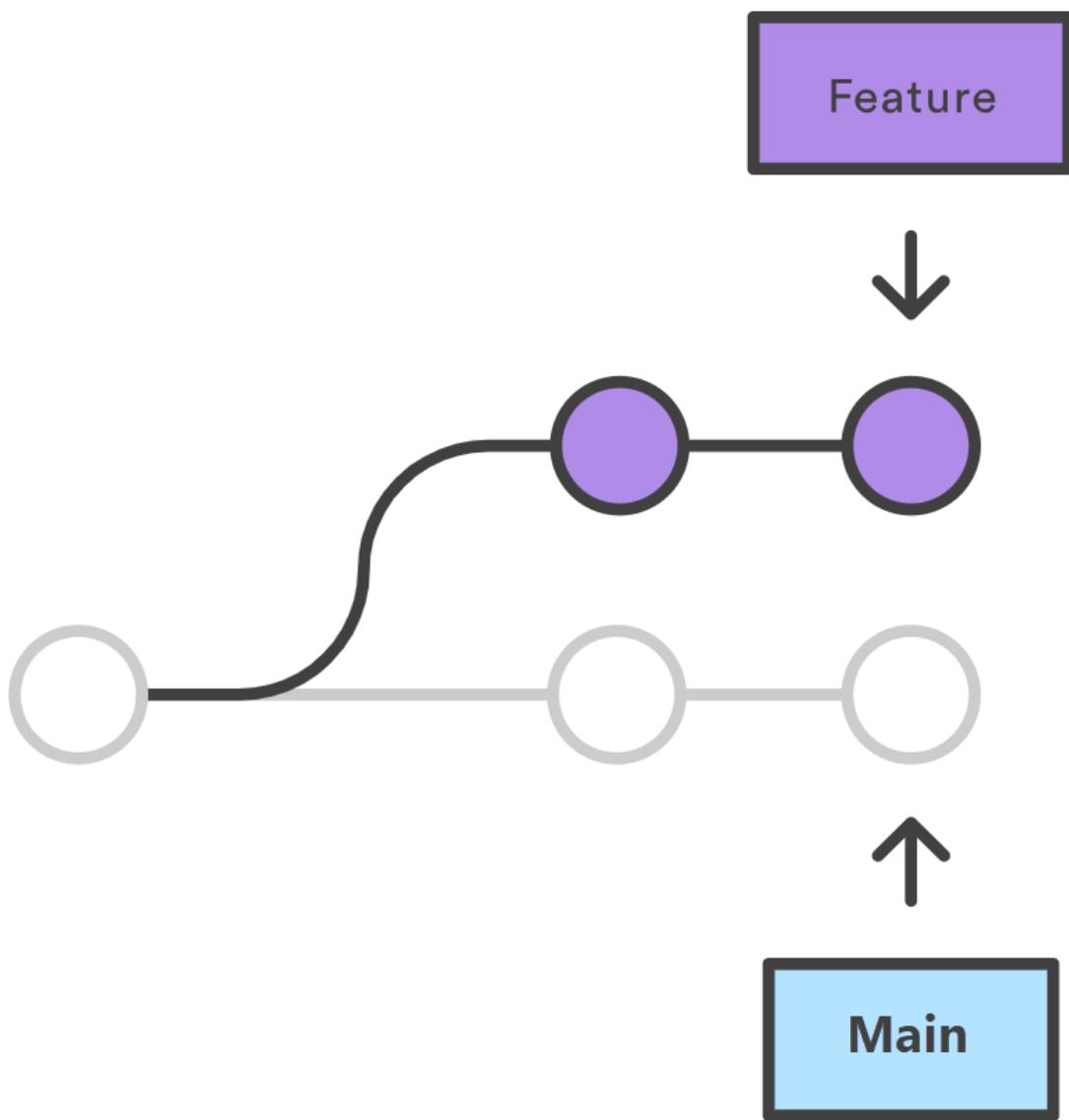
Having a complete local history makes Git fast since you do not need a network connection to create commits, inspect previous versions of a file, or do diffs between commits.

Distributed development also makes it easier to scale your engineering team. If someone breaks the production branch in SVN, other developers cannot check in their changes until it is fixed. With Git, this kind of blocking does not exist. Everybody can continue going about their business in their local repositories.

And, like feature branches, distributed development creates a more reliable environment. Even if developers obliterate their repository, they can clone from someone else and start afresh.

## Trunk-based development

One of the most significant advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge.



Trunk-based development provides an isolated environment for every change to your codebase. When developers want to start working on something—no matter how large or small—they create a new branch. It ensures that the main branch always contains production-quality code.

Using trunk-based development is more reliable than directly-editing production code, but it also provides organizational benefits.

They let you represent development work at the same granularity as your agile backlog.

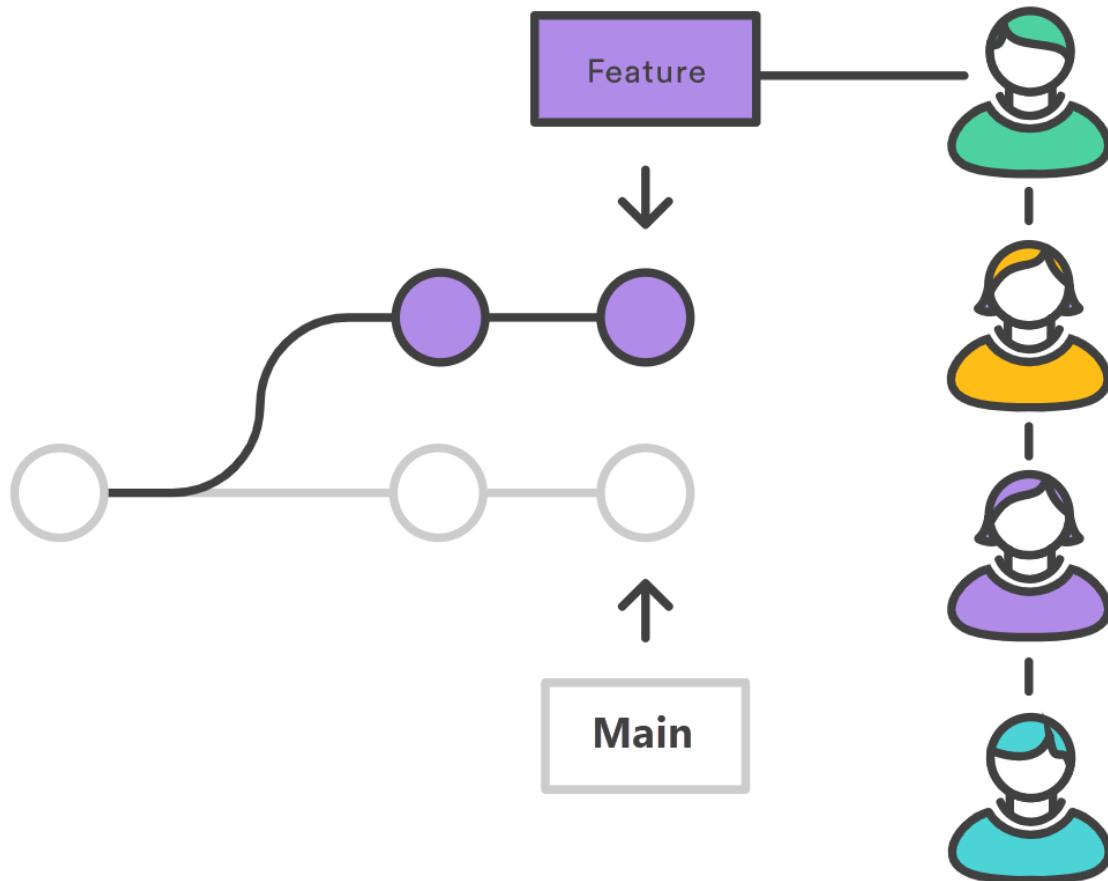
For example, you might implement a policy where each work item is addressed in its feature branch.

## Pull requests

Many source code management tools such as Azure Repos enhance core Git functionality with pull requests.

A pull request is a way to ask another developer to merge one of your branches into their repository.

It makes it easier for project leads to keep track of changes and lets developers start discussions around their work before integrating it with the rest of the codebase.



Since they are essentially a comment thread attached to a feature branch, pull requests are incredibly versatile.

When a developer gets stuck with a complex problem, they can open a pull request to ask for help from the rest of the team.

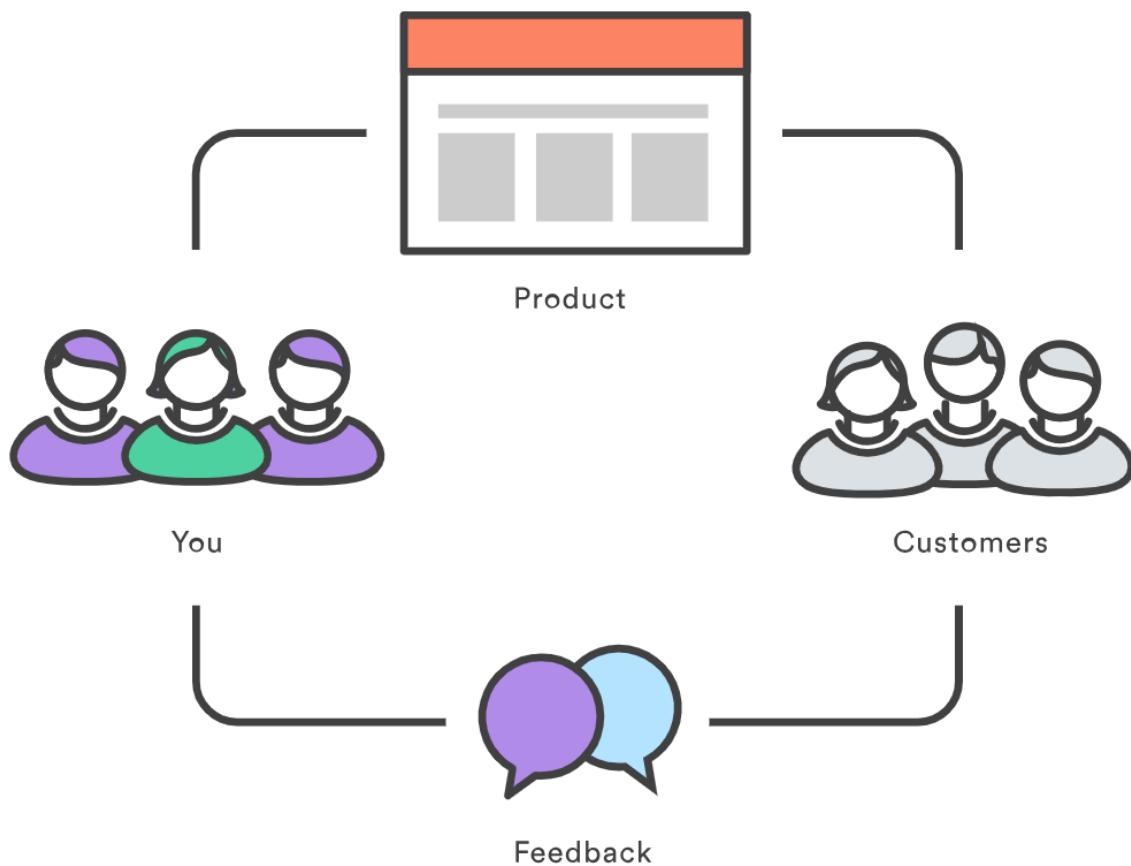
Instead, junior developers can be confident that they are not destroying the entire project by treating pull requests as a formal code review.

## Faster release cycle

A faster release cycle is the ultimate result of feature branches, distributed development, pull requests, and a stable community.

These capabilities promote an agile workflow where developers are encouraged to share more minor changes more frequently.

In turn, changes can get pushed down the deployment pipeline faster than the standard of the monolithic releases with centralized version control systems.



As you might expect, Git works well with continuous integration and continuous delivery environments.

Git hooks allow you to run scripts when certain events occur inside a repository, which lets you automate deployment to your heart's content.

You can even build or deploy code from specific branches to different servers.

For example, you might want to configure Git to deploy the most recent commit from the develop branch to a test server whenever anyone merges a pull request into it.

Combining this kind of build automation with peer review means you have the highest possible confidence in your code as it moves from development to staging to production.

## Understand objections to using Git

Completed 100 XP

- 2 minutes

There are three common objections I often hear to migrating to Git:

- I can overwrite history.
- I have large files.
- There is a steep learning curve.

### Overwriting history

Git technically does allow you to overwrite history - but like any helpful feature, if misused can cause conflicts.

If your teams are careful, they should never have to overwrite history.

If you are synchronizing to Azure Repos, you can also add a security rule that prevents developers from overwriting history by using the explicit "Force Push" permissions.

Every source control system works best when developers understand how it works and which conventions work.

While you cannot overwrite history with Team Foundation Version Control (TFVC), you can still overwrite code and do other painful things.

## Large files

Git works best with repos that are small and do not contain large files (or binaries).

Every time you (or your build machines) clone the repo, they get the entire repo with its history from the first commit.

It is great for most situations but can be frustrating if you have large files.

Binary files are even worse because Git cannot optimize how they are stored.

That is why [Git LFS](#) was created.

It lets you separate large files of your repos and still has all the benefits of versioning and comparing.

Also, if you are used to storing compiled binaries in your source repos, stop!

Use [Azure Artifacts](#) or some other package management tool to store binaries for which you have source code.

However, teams with large files (like 3D models or other assets) can use Git LFS to keep the code repo slim and trimmed.

## Learning curve

There is a learning curve. If you have never used source control before, you are probably better off when learning Git. I have found that users of centralized source control (TFVC or SubVersion) battle initially to make the mental shift, especially around branches and synchronizing.

Once developers understand how Git branches work and get over the fact that they must commit and then push, they have all the basics they need to succeed in Git.

## Describe working with Git locally

Completed 100 XP

- 7 minutes

Git and Continuous Delivery is one of those delicious chocolate and peanut butter combinations. We occasionally find two great tastes that taste great together in the software world!

Continuous Delivery of software demands a significant level of automation. It is hard to deliver continuously if you do not have a quality codebase.

Git provides you with the building blocks to take charge of quality in your codebase. It allows you to automate most of the checks in your codebase.

Also, it works before committing the code into your repository.

To fully appreciate the effectiveness of Git, you must first understand how to carry out basic operations on Git. For example, clone, commit, push, and pull.

The natural question is, how do we get started with Git?

One option is to go native with the command line or look for a code editor that supports Git natively.

Visual Studio Code is a cross-platform, open-source code editor that provides powerful developer tooling for hundreds of languages.

To work in open-source, you need to embrace open-source tools.

This recipe will start by:

- Setting up the development environment with Visual Studio Code.
- Creating a new Git repository.
- Committing code changes locally.
- Pushing changes to a remote repository on Azure DevOps.

## Getting ready

This tutorial will teach us how to initialize a Git repository locally.

Then we will use the ASP.NET Core MVC project template to create a new project and version it in the local Git repository.

We will then use Visual Studio Code to interact with the Git repository to do basic commit, pull, and push operations.

You will need to set up your working environment with the following:

- .NET Core 3.1 SDK or later: [Download .NET](#).
- Visual Studio Code: [Download Visual Studio Code](#).
- C# Visual Studio Code extension: [C# programming with Visual Studio Code](#).
- Git: [Git - Downloads](#)
- Git for Windows (if you are using Windows): [Git for Windows](#)

The Visual Studio Marketplace features several extensions for Visual Studio Code that you can install to enhance your experience of using Git:

- [Git Lens](#): This extension brings visualization for code history by using Git blame annotations and code lens. The extension enables you to seamlessly navigate and explore the history of a file or branch. Also, the extension allows you to gain valuable insights via powerful comparison commands and much more.
- [Git History](#): Brings visualization and interaction capabilities to view the Git log, file history and compare branches or commits.

## How to do it

1. Open the Command Prompt and create a new-working folder:

```
CmdCopy  
mkdir myWebApp  
cd myWebApp
```

2. In myWebApp, initialize a new Git repository:

```
CmdCopy  
git init
```

3. Configure global settings for the name and email address to be used when committing in this Git repository:

```
CmdCopy  
git config --global user.name "John Doe"  
git config --global user.email "john.doe@contoso.com"
```

If you are working behind an enterprise proxy, you can make your Git repository proxy-aware by adding the proxy details in the Git global configuration file.

Different variations of this command will allow you to set up an HTTP/HTTPS proxy (with username/password) and optionally bypass SSL verification.

Run the below command to configure a proxy in your global git config.

CmdCopy

```
git config --global http.proxy  
http://proxyUsername:proxyPassword@proxy.server.com:port
```

4. Create a new ASP.NET core application. The new command offers a collection of switches that can be used for language, authentication, and framework selection. More details can be found on [Microsoft docs](#).

CmdCopy

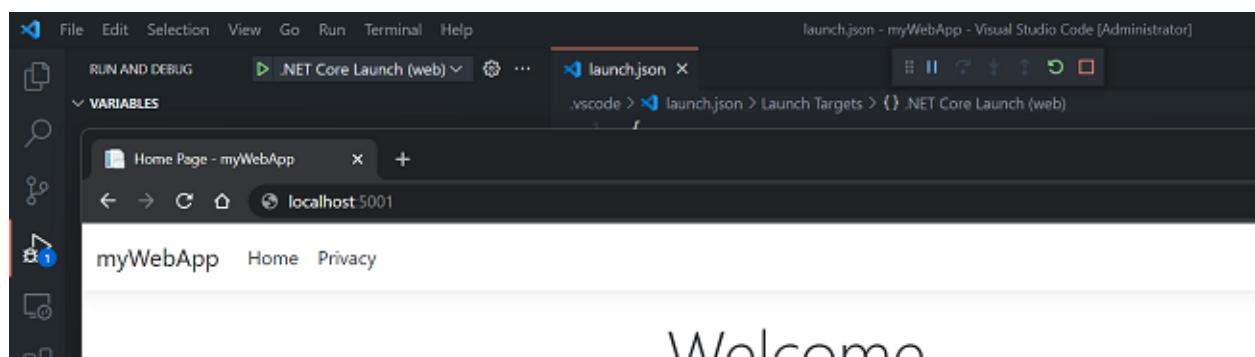
```
dotnet new mvc
```

Launch Visual Studio Code in the context of the current-working folder:

CmdCopy

```
code .
```

5. When the project opens in Visual Studio Code, select **Yes** for the **Required assets to build and debug are missing from 'myWebApp.'** **Add them?** Warning message. Select **Restore** for the **There are unresolved dependencies** info message. Hit **F5** to debug the application, then myWebApp will load in the browser, as shown in the following screenshot:

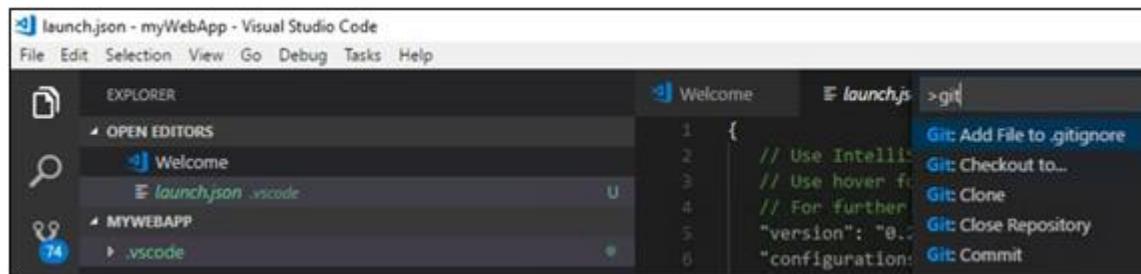


If you prefer to use the command line, you can run the following commands in the context of the git repository to run the web application.

### CmdCopy

```
dotnet build  
dotnet run
```

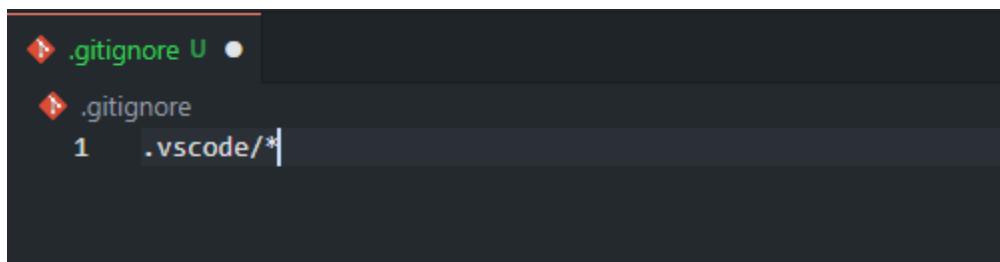
You will notice the ".vscode" folder is added to your working folder. To avoid committing this folder to your Git repository, you can include it in the .gitignore file. Select a file from the ".vscode" folder, hit F1 to launch the command window in Visual Studio Code, type gitignore, and accept the option to include the selected file in the new .gitignore file.



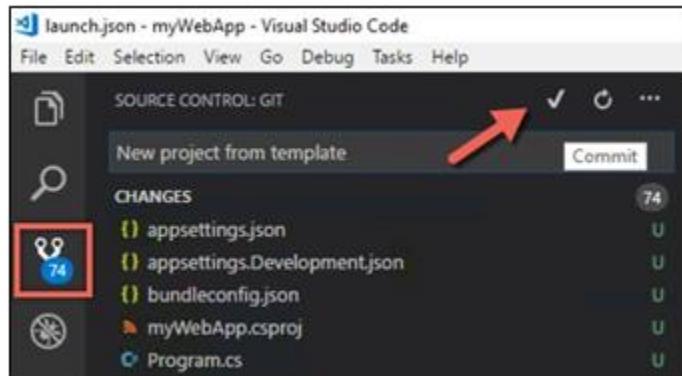
### Note

To ignore an entire directory, you need to include the name of the directory with the slash / at the end.

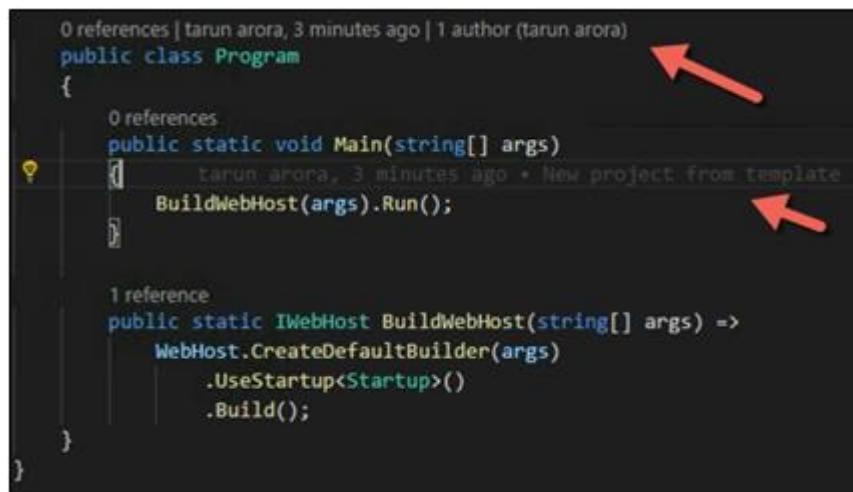
Open your .gitignore, remove the file name from the path, and leave the folder with a slash, for example, .vscode/\*.



6. To stage and commit the newly created myWebApp project to your Git repository from Visual Studio Code, navigate the Git icon from the left panel. Add a commit comment and commit the changes by clicking the checkmark icon. It will stage and commit the changes in one operation:



Open Program.cs, you will notice Git lens decorates the classes and functions with the commit history and brings this information in line to every line of code:



7. Now launch cmd in the context of the git repository and run `git branch --list`. It will show you that only the `main` branch currently exists in this repository. Now run the following command to create a new branch called `feature-devops-home-page`.

#### CmdCopy

```
git branch feature-devops-home-page  
git checkout feature-devops-home-page  
git branch --list
```

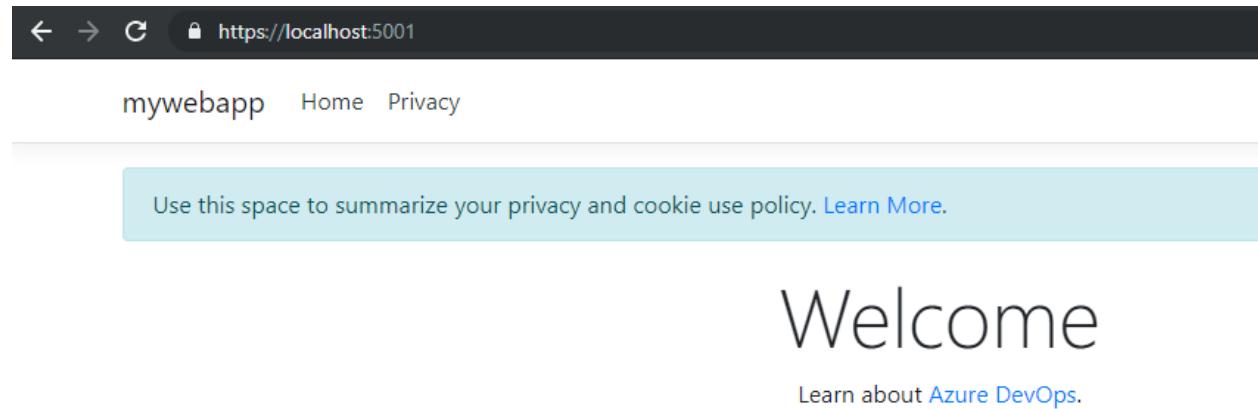
You have created a new branch with these commands and checked it out. The `--list` keyword shows you a list of all branches in your repository. The green color represents the branch that is currently checked out.

- Now navigate to the file ~\Views\Home\Index.cshtml and replace the contents with the text below.

```
C#Copy
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://azure.microsoft.com/services/devops/">Azure DevOps</a>.</p>
</div>
```

- Refresh the web app in the browser to see the changes.



- In the context of the git repository, execute the following commands. These commands will stage the changes in the branch and then commit them.

```
CmdCopy
git status

git add .

git commit -m "updated welcome page."

git status
```

- To merge the changes from the feature-devops-home-page into the main, run the following commands in the context of the git repository.

```
CmdCopy
```

```
git checkout main  
git merge feature-devops-home-page
```

```
Updating 5d2441f..e9c9484  
Fast-forward  
 Views/Home/Index.cshtml | 4 +--  
 1 file changed, 2 insertions(+), 2 deletions(-)
```

12. Run the below command to delete the feature branch.

CmdCopy

```
git branch --delete feature-devops-home-page
```

## How it works

The easiest way to understand the outcome of the steps done earlier is to check the history of the operation. Let us have a look at how to do it.

1. In Git, committing changes to a repository is a two-step process.  
Running: `add .` The changes are staged but not committed. Finally, running the commit promotes the staged changes in the repository.
2. To see the history of changes in the main branch, run the command `git log -v`

```
commit e9c948427c1aa99e8aede67f6a2be206d148beaf  
Author: Tarun Arora <tarun.arora@contoso.com>  
Date: Thu Jul 25 12:45:43 2019 +0100  
  
    updated welcome page  
  
commit 5d2441f0be4f1e4ca1f8f83b56dee31251367adc  
Author: Tarun Arora <tarun.arora@contoso.com>  
Date: Thu Jul 25 12:07:55 2019 +0100  
  
    project init
```

3. To investigate the actual changes in the commit, you can run the command `git log -p`

```
commit e9c948427c1aa99e8aede67f6a2be206d148beaf
Author: Tarun Arora <tarun.arora@contoso.com>
Date:   Thu Jul 25 12:45:43 2019 +0100

    updated welcome page

diff --git a/Views/Home/Index.cshtml b/Views/Home/Index.cshtml
index d2d19bd..6d8ad94 100644
--- a/Views/Home/Index.cshtml
+++ b/Views/Home/Index.cshtml
@@ -4,5 +4,5 @@
<div class="text-center">
    <h1 class="display-4">Welcome</h1>
-   <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
-</div>
+   <p>Learn about <a href="https://azure.microsoft.com/en-gb/services/devops/">Azure DevOps</a>.</p>
+</div>
\ No newline at end of file
```

## There is more

Git makes it easy to back out changes. Following our example, if you want to take out the changes made to the welcome page.

You can do it hard resetting the main branch to a previous version of the commit using the following command.

```
CmdCopy
git reset --hard 5d2441f0be4f1e4ca1f8f83b56dee31251367adc
```

Running the above command would reset the branch to the project init change.

If you run `git log -v`, you will see that the changes done to the welcome page are removed from the repository.

# Work with Azure Repos and GitHub

## Introduction to Azure Repos

Completed 100 XP

- 2 minutes

Azure Repos is a set of version control tools that you can use to manage your code.

Using version control is a good idea whether your software project is large or small.

Azure Repos provides two types of version control:

- Git: distributed version control
- Team Foundation Version Control (TFVC): centralized version control

## What do I get with Azure Repos?

- Use free private Git repositories, pull requests, and code search: Get unlimited private Git repository hosting and support for TFVC that scales from a hobby project to the world's largest repository.
- Support for any Git client: Securely connect with and push code into your Git repository from any IDE, editor, or Git client.
- Web hooks and API integration: Add validations and extensions from the marketplace or build your own using web hooks and REST APIs.
- Semantic code search: Quickly find what you are looking for with a code-aware search that understands classes and variables.
- Collab to build better code: Do more effective Git code reviews with threaded discussion and continuous integration for each change. Use forks to promote collaboration with inner source workflows.
- Automation with built-in CI/CD: Set up continuous integration/continuous delivery (CI/CD) to automatically trigger builds, tests, and deployments. Including every completed pull request using Azure Pipelines or your tools.
- Protection of your code quality with branch policies: Keep code quality high by requiring code reviewer sign-off, successful builds, and passing tests before merging pull requests. Customize your branch policies to maintain your team's high standards.
- Usage of your favorite tools: Use Git and TFVC repositories on Azure Repos with your favorite editor and IDE.

For further reference on using git in Azure Repos, refer to [Microsoft Docs](#).

# Introduction to GitHub

Completed 100 XP

- 3 minutes

GitHub is the largest open-source community in the world. Microsoft owns GitHub. GitHub is a development platform inspired by the way you work.

You can host and review code, manage projects, and build software alongside 40 million developers from open source to business.

GitHub is a Git repository hosting service that adds many of its features.

While Git is a command-line tool, GitHub provides a Web-based graphical interface.

It also provides access control and several collaboration features, such as wikis and essential task management tools for every project.

So what are the main benefits of using GitHub? Nearly every open-source project uses GitHub to manage its project.

Using GitHub is free if your project is open source and includes a wiki and issue tracker, making it easy to have more in-depth documentation and get feedback about your project.

## What are some of the features offered by GitHub?

- Automate from code to cloud: Cycle your production code faster and simplify your workflow with GitHub Packages and built-in CI/CD using GitHub Actions.
  - Automate your workflows: Build, test, deploy, and run CI/CD how you want in the same place you manage code. Trigger Actions from any GitHub event to any available API. Build your Actions in the language of your choice, or choose from thousands of workflows and Actions created by the community.
  - Packages at home with their code: Use Actions to automatically publish new package versions to GitHub Packages. Install

packages and images hosted on GitHub Packages or your preferred packages registry in your CI/CD workflows. It is always free for open source, and data transfer within Actions is unlimited for everyone.

- Securing software together: GitHub plays a role in securing the world's code—developers, maintainers, researchers, and security teams. On GitHub, development teams everywhere can work together to secure the world's software supply chain, from fork to finish.
  - **Get alerts about vulnerabilities in your code:** GitHub continuously scans security advisories for popular languages. Also, it sends security alerts to maintainers of affected repositories with details so they can remediate risks.
  - **Automatically update vulnerabilities:** GitHub monitors your project dependencies and automatically opens pull requests to update dependencies to the minimum version that resolves known vulnerabilities.
  - **Stay on top of CVEs:** Stay updated with the latest Common Vulnerabilities and Exposures (CVEs), and learn how they affect you with the GitHub Advisory Database.
  - Find vulnerabilities that other tools miss: CodeQL is the industry's leading semantic code analysis engine. GitHub's revolutionary approach treats code as data to identify security vulnerabilities faster.
  - Eliminate variants: Never make the same mistake twice. Proactive vulnerability scanning prevents vulnerabilities from ever reaching production.
  - Keep your tokens safe: Accidentally commit a token to a public repository? GitHub got you. With support from 20 service providers, GitHub takes steps to keep you safe.
- Seamless code review: Code review is the surest path to better code and is fundamental to how GitHub works. Built-in review tools make code review an essential part of your team's process.
  - Propose changes: Better code starts with a Pull Request, a living conversation about changes where you can talk through ideas, assign tasks, discuss details, and conduct reviews.
  - Request reviews: If you are on the other side of a review, you can request reviews from your peers to get the detailed feedback you need.

- See the difference: Reviews happen faster when you know exactly what changes. Diffs compare versions of your source code, highlighting the new, edited, or deleted parts.
  - Comment in context: Discussions happen in comment threads within your code—bundle comments into one review or reply to someone else who is in line to start a conversation.
  - Give clear feedback: Your teammates should not have to think too hard about what a thumbs-up emoji means. Specify whether your comments are required changes or just a few suggestions.
  - Protect branches: Only merge the highest-quality code. You can configure repositories to require status checks, reducing human error and administrative overhead.
- All your code and documentation in one place: Hundreds of millions of private, public, and open-source repositories are hosted on GitHub. Every repository has tools to help you host, version, and release code and documentation.
  - Code where you collaborate: Repositories keep code in one place and help your teams collaborate with the tools they love, even if you work with large files using Git LFS. You can create or import as many projects as possible with unlimited private repositories for individuals and groups.
  - Documentation alongside your code: Host your documentation directly from your repositories with GitHub Pages. Use Jekyll as a static site generator and publish your Pages from the /docs folder on your main branch.
- Manage your ideas: Coordinate early, stay aligned, and get more done with GitHub's project management tools.
  - See your project's large picture: See everything happening in your project and choose where to focus your team's efforts with Projects and task boards that live right where they belong: close to your code.
  - Track and assign tasks: Issues help you identify, assign, and keep track of tasks within your team. You can open an Issue to track a bug, discuss an idea with an @mention, or start distributing work.
- The human side of software: Building software is more about managing teams and communities than coding. Whether on a group of two or 2000, GitHub has the support your people need.
  - Manage and grow teams: Help people organize with GitHub teams, level up to access administrative roles, and fine-tune your permissions with nested teams.

- Keep conversations: Moderation tools, like issue and pull request locking, help your team stay focused on code. And if you maintain an open-source project, user blocking reduces noise and ensures productive conversations.
- Set community guidelines: Set roles and expectations without starting from scratch. Customize standard codes of conduct to create the perfect one for your project. Then choose a pre-written license right from your repository.

GitHub offers excellent learning resources for its platform. You can find everything from git introduction training to deep dive on publishing static pages to GitHub and how to do DevOps on GitHub right [here](#).

## Migrate from TFVC to Git

Completed 100 XP

- 2 minutes

### Migrating the tip

Most teams wish they could reorganize their source control structure.

Typically, the structure the team is using today was set up by a well-meaning developer a decade ago, but it is not optimal.

Migrating to Git could be an excellent opportunity to restructure your repo.

In this case, it probably does not make sense to migrate history anyway since you are going to restructure the code (or break the code into multiple repos).

The process is simple:

- Create an empty Git repo (or multiple empty repos).
- Get-latest from TFS.
- Copy/reorganize the code into the empty Git repos.
- Commit and push, and you are there!

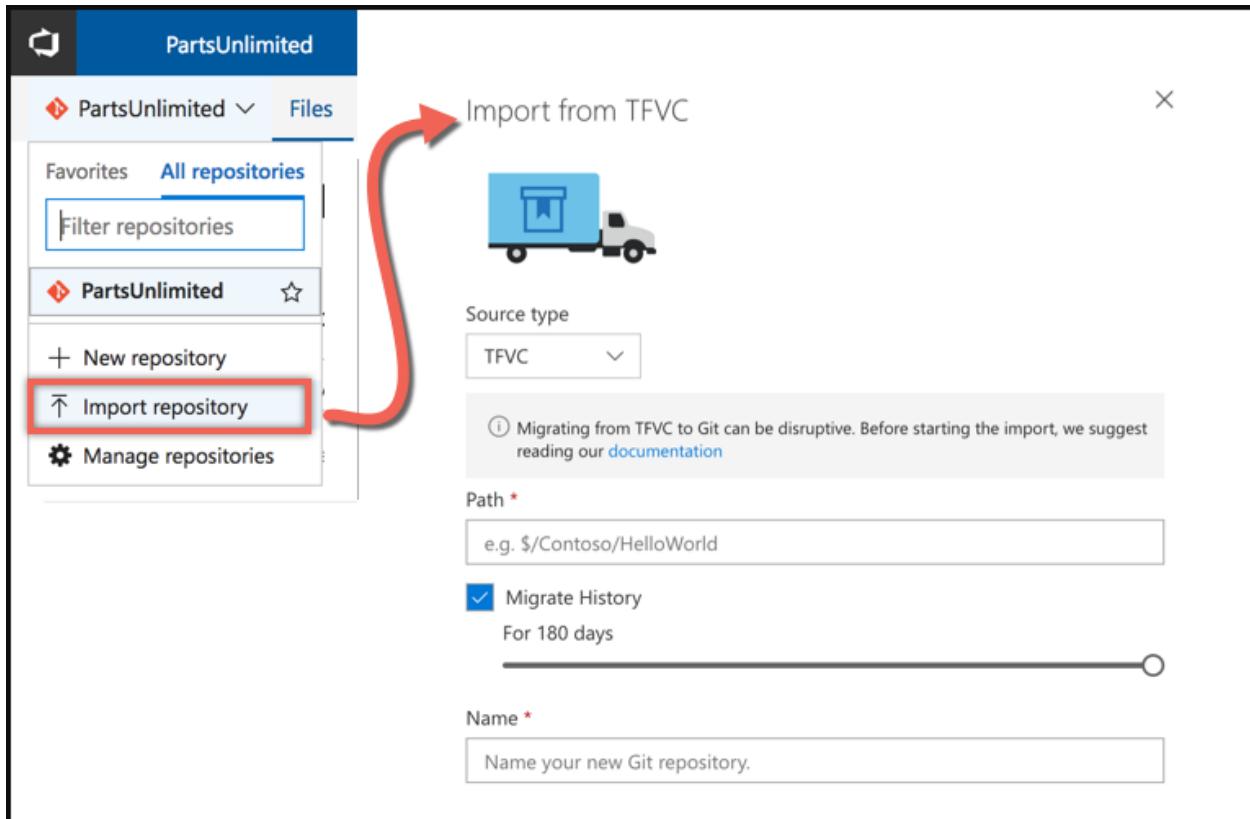
If you have shared code, you need to create builds of the shared code to publish to a package feed. And then consume those packages in downstream applications, but the Git part is straightforward.

## Single branch import

If you are on TFVC and in Azure DevOps, you have the option of a simple single-branch import. Click on the Import repository from the Azure Repos top-level drop-down menu to open the dialog. Then enter the path to the branch you are migrating to (yes, you can only choose one branch). Select if you want history or not (up to 180 days). Add in a name for the repo, and the import will be triggered.

## Import repository

Import repository also allows you to import a git repository. It is beneficial to move your git repositories from GitHub or any other public or private hosting spaces into Azure Repos.



There are some limitations here (that apply only when migrating source type TFVC): a single branch and only 180 days of history.

However, if you only care about one branch and are already in Azure DevOps, it is an effortless but effective way to migrate.

## Use GIT-TFS

Completed 100 XP

- 3 minutes

What if you need to migrate more than a single branch and keep branch relationships? Or are you going to drag all the history with you?

In that case, you're going to have to use **GIT-TFS**. It's an open-source project built to synchronize Git and TFVC repositories.

But you can do a once-off migration using the Git TFS clone.

**GIT-TFS** has the advantage that it can migrate multiple branches and preserve the relationships to merge branches in Git after you migrate.

Be warned that doing this conversion can take a while - especially for large or long-history repositories.

You can quickly dry-run the migration locally, iron out any issues, and then do it for real. There are lots of flexibilities with this tool.

If you are on Subversion, you can use **GIT-SVN** to import your Subversion repo similarly to **GIT-TFS**.

## Migrating from TFVC to Git using GIT-TFS

If Chocolatey is already installed on your computer, run `choco install gittfs`

Add the GIT-TFS folder path to your PATH. You could also set it temporary (the time of your current terminal session) using: `set PATH=%PATH%;%cd%\GitTfs\bin\Debug`

You need .NET 4.5.2 and maybe the 2012 or 2013 version of Team Explorer (or Visual Studio). It depends on the version of Azure DevOps you want to target.

Clone the whole repository (wait for a while.) :

```
git tfs clone http://tfs:8080/tfs/DefaultCollection $/some_project
```

Advanced use cases of cloning the TFVC repository into Git are [documented here](#).

CmdCopy  
cd some\_project  
git log

## Develop online with GitHub Codespaces

Completed 100 XP

- 2 minutes

GitHub Codespaces addresses several issues from which developers regularly suffer.

First, many developers are working with old hardware and software systems, which are not refreshed.

Second, developers are often tied to individual development systems. Moving from location to location or system to system is inconvenient or slow to configure.

A problem for the developers' organizations is the proliferation of intellectual property across all these machines.

## What is GitHub Codespaces?

Codespaces is a cloud-based development environment that GitHub hosts. It is essentially an online implementation of Visual Studio Code.

Codespaces allows developers to work entirely in the cloud.

Codespaces even will enable developers to contribute from tablets and Chromebooks.

Because it is based on Visual Studio Code, the development environment is still rich with:

- Syntax highlighting.
- Autocomplete.
- Integrated debugging.
- Direct Git integration.

Developers can create a codespace (or multiple codespaces) for a repository. Each codespace is associated with a specific branch of a repository.

## Using GitHub Codespaces

You can do all your work in codespaces within a browser.

For an even more responsive experience, you can connect to a codespace from a local copy of Visual Studio Code.

# Structure your Git Repo

## Explore monorepo versus multiple repos

Completed 100 XP

- 3 minutes

A repository is where your work history is stored, usually in a git subdirectory.

How should you organize your code repository? Development teams aim to separate concerns in their software and repositories. As time passes, it isn't unusual for code repositories to become cluttered with irrelevant code and artifacts.

When it comes to organizing your repositories, there are two main philosophies: using a single repository (Monorepo) or multiple repositories.

- Monorepos is a source control pattern where all source code is kept in one repository. It's easy to give all employees access to everything at once. Clone it, and you're done.
- Organizing your projects into separate repositories is referred to as multiple repositories.

The fundamental difference between mono repo and multiple repo philosophies is what enables teams to work together most efficiently. In an extreme scenario, the multiple repos view suggests that each subteam can work in its repository. It allows them to work in their respective areas using the libraries, tools, and development workflows that optimize their productivity.

The cost of consuming anything not developed within a given repository is equivalent to using a third-party library or service, even if it was written by someone sitting nearby.

If you come across a bug in your library, you should address it in the corresponding repository. Once you have published a new artifact, you can return to your repository and make the necessary code changes. However, if the bug is in a different code base or involves different libraries, tools, or workflows, you may need to seek assistance from the owner of that system and wait for their response.

When using the mono repo view, managing complex dependency graphs can increase the difficulty of using a single repository. The benefits of allowing different teams to work independently aren't substantial. Some teams may find an efficient way of working,

but this may not be true for all groups. Furthermore, other teams may choose a suboptimal approach, negating any benefits gained by others. Consolidating all your work in a mono repo lets you focus on closely monitoring this single repository.

The hassle of making changes in other repos or waiting for teams to make changes for you is avoided in a mono repo where anyone can change anything.

If you discover a bug in a library, fixing it's as easy as finding a bug in your own code.

### Note

In Azure DevOps, it's common to use a separate repository for each associated solution within a project.

## Implement a change log

Completed 100 XP

- 3 minutes

The concept of a changelog is simple enough: It's a file that has a list of changes made to a project, usually in date order. The typical breakdown is to separate a list of versions, and then within each version, show:

- Added features
- Modified/Improved features
- Deleted features

Some teams will post changelogs as blog posts; others will create a CHANGELOG.md file in a GitHub repository.

## Automated change log tooling

While changelogs can be created and manually maintained, you might want to consider using an automated changelog creation tool. At least as a starting point.

## Using native GitHub commands

The git log command can be useful for automatically creating content. Example: create a new section per version:

BashCopy

```
git log [options] vX.X.X..vX.X.Y | helper-script > projectchangelogs/X.X.Y
```

## Git changelog

One standard tool is [gitchangelog](#). This tool is based on Python.

## GitHub changelog generator

Another standard tool is called [github-changelog-generator](#).

BashCopy

```
$ github_changelog_generator -u github-changelog-generator -p TimerTrend-3.0
```

This tool is based on Gem.

## Should you use autogenerated log-based data?

Preference is always to avoid dumping log entries into a changelog. Logs are "noisy," so it's easy to generate a mess that isn't helpful.

# Manage Git branches and workflows

## Explore branch workflow types

Completed 100 XP

- 4 minutes

### What is a successful Git branch workflow?

When evaluating a workflow for your team, you must consider your team's culture. You want the workflow to enhance your team's effectiveness and not be a burden that limits productivity. Some things to consider when evaluating a Git workflow are:

- Does this workflow scale with team size?
- Is it easy to undo mistakes and errors with this workflow?
- Does this workflow impose any new unnecessary cognitive overhead on the team?

### Common branch workflows

Most popular Git workflows will have some sort of centralized repo that individual developers will push and pull from.

Below is a list of some popular Git workflows that we'll go into more detail about in the next section.

These comprehensive workflows offer more specialized patterns about managing branches for feature development, hotfixes, and eventual release.

### Trunk-based development

Trunk-based development is a logical extension of Centralized Workflow.

The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the main branch.

This encapsulation makes it easy for multiple developers to work on a particular feature without disturbing the main codebase.

It also means the main branch should never contain broken code, which is a huge advantage for continuous integration environments.

## Forking workflow

The Forking Workflow is fundamentally different than the other workflows discussed in this tutorial.

Instead of using a single server-side repository to act as the "central" codebase, it gives every developer a server-side repository.

It means that each contributor has two Git repositories:

- A private local one.
- A public server-side one.

## Explore feature branch workflow

Completed 100 XP

- 4 minutes

The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the main branch.

The encapsulation makes it easy for multiple developers to work on a particular feature without disturbing the main codebase. It also means the main branch will never contain broken code, a huge advantage for continuous integration environments.

Encapsulating feature development also makes it possible to use pull requests, which are a way to start discussions around a branch. They allow other developers to sign off on a feature before it integrates into the official project. Or, if you get stuck in the middle of a feature, you can open a pull request asking for suggestions from your colleagues.

Pull requests make it incredibly easy for your team to comment on each other's work. Also, feature branches can (and should) be pushed to the central repository. It allows sharing a feature with other developers without touching any official code.

Since the main is the only "special" branch, storing several feature branches on the central repository doesn't pose any problems. It's also a convenient way to back up everybody's local commits.

## Trunk-based development workflow

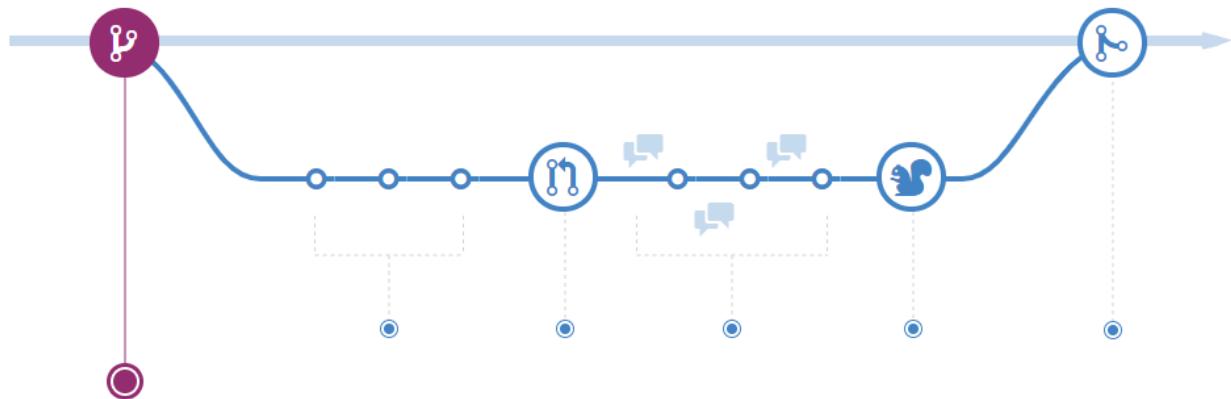
The trunk-based development Workflow assumes a central repository, and the main represents the official project history.

Instead of committing directly to their local main branch, developers create a new branch whenever they start working on a new feature.

Feature branches should have descriptive names, like new-banner-images or bug-91. The idea is to give each branch a clear, highly focused purpose.

Git makes no technical distinction between the main and feature branches, so developers can edit, stage, and commit changes to a feature branch.

### Create a branch



When you're working on a project, you will have many different features or ideas in progress at any given time – some of which are ready to go and others that aren't.

Branching exists to help you manage this workflow.

When you create a branch in your project, you create an environment where you can try out new ideas.

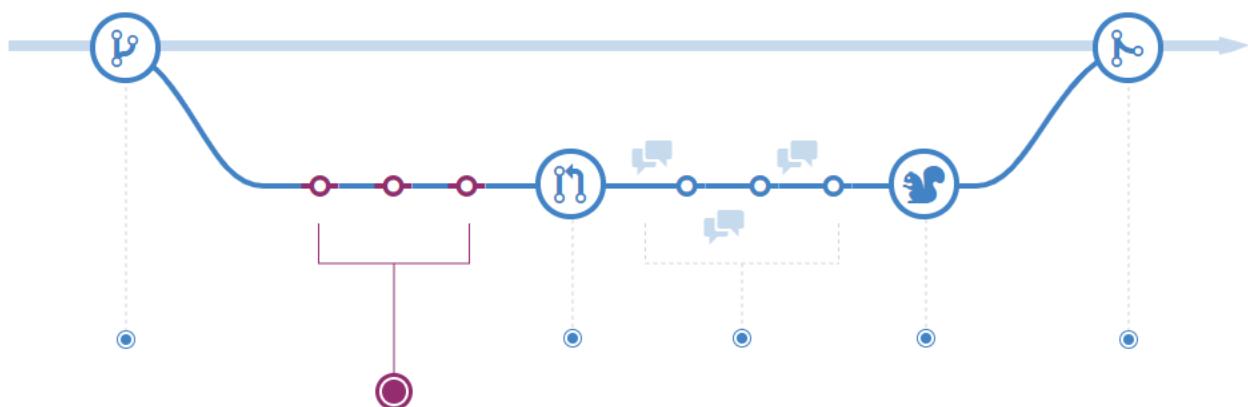
Changes you make on a branch don't affect the main branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

Branching is a core concept in Git; the entire branch flow is based upon it. Only one rule: anything in the main branch is always deployable.

Because of this, your new branch must be created off the main when working on a feature or a fix.

Your branch name should be descriptive (for example, refactor-authentication, user-content-cache-key, make-retina-avatars) so others can see what is being worked on.

## Add commits



Once your branch has been created, it's time to make changes. Whenever you add, edit, or delete a file, you make a commit and add them to your branch.

Adding commits keeps track of your progress as you work on a feature branch.

Commits also create a transparent history of your work that others can follow to understand your actions and why.

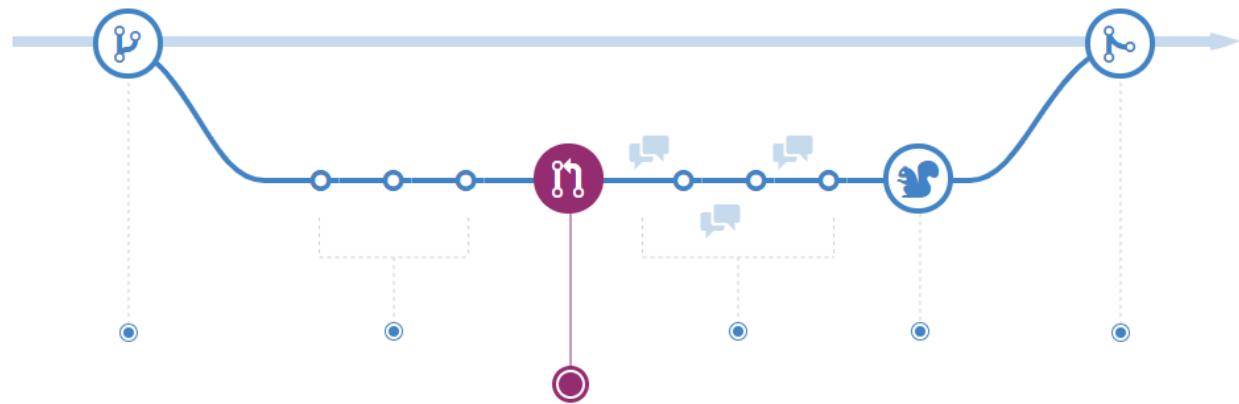
Each commit has an associated commit message explaining why a particular change was made.

Furthermore, each commit is considered a separate unit of change. It lets you roll back changes if a bug is found or you decide to head in a different direction.

Commit messages are essential, especially since Git tracks your changes and displays them as commits once pushed to the server.

By writing clear commit messages, you can make it easier for others to follow along and provide feedback.

## Open a pull request



The Pull Requests start a discussion about your commits. Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.

You can open a Pull Request at any point during the development process when:

- You've little or no code but want to share screenshots or general ideas.
- You're stuck and need help or advice.
- You're ready for someone to review your work.

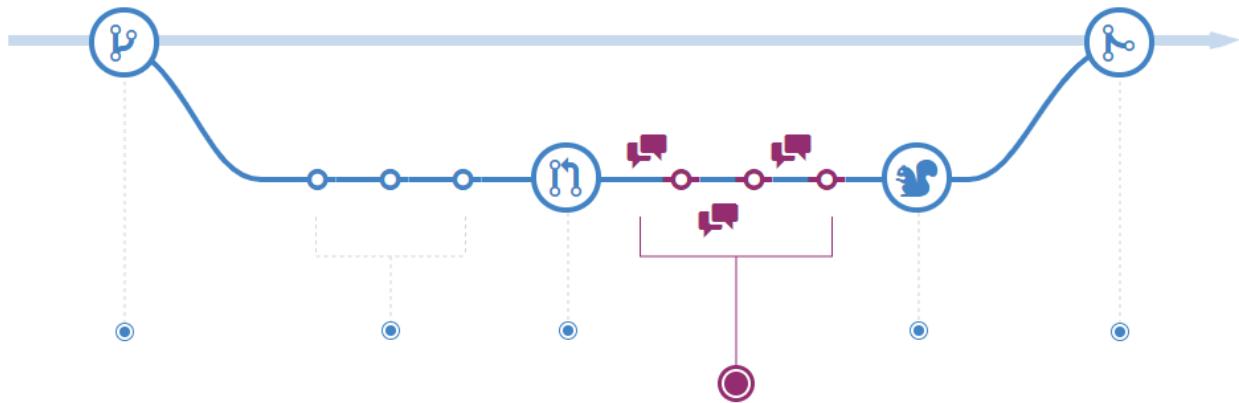
Using the @mention system in your Pull Request message, you can ask for feedback from specific people or teams, whether they're down the hall or 10 time zones away.

Pull Requests help contribute to projects and for managing changes to shared repositories.

If you're using a Fork & Pull Model, Pull Requests provide a way to notify project maintainers about the changes you'd like them to consider.

If you're using a Shared Repository Model, Pull Requests help start code review and conversation about proposed changes before they're merged into the main branch.

## Discuss and review your code



Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments.

Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, everything looks excellent, and the props are in order.

Pull Requests are designed to encourage and capture this type of conversation.

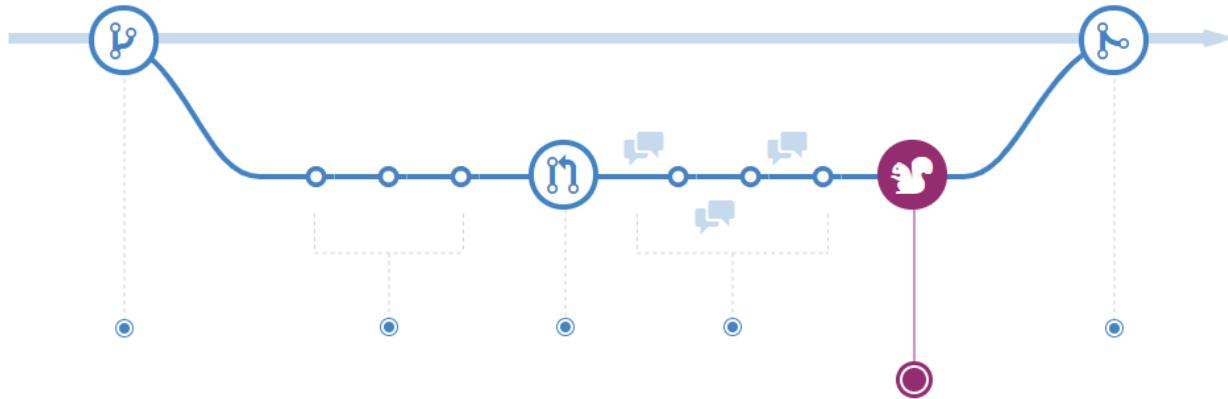
You can also continue to push to your branch, considering discussion and feedback about your commits.

Suppose someone comments that you forgot to do something, or if there's a bug in the code, you can fix it in your branch and push up the change.

Git will show your new commits and any feedback you may receive in the unified Pull Request view.

Pull Request comments are written in Markdown, so you can embed images and emojis, use pre-formatted text blocks, and other lightweight formatting.

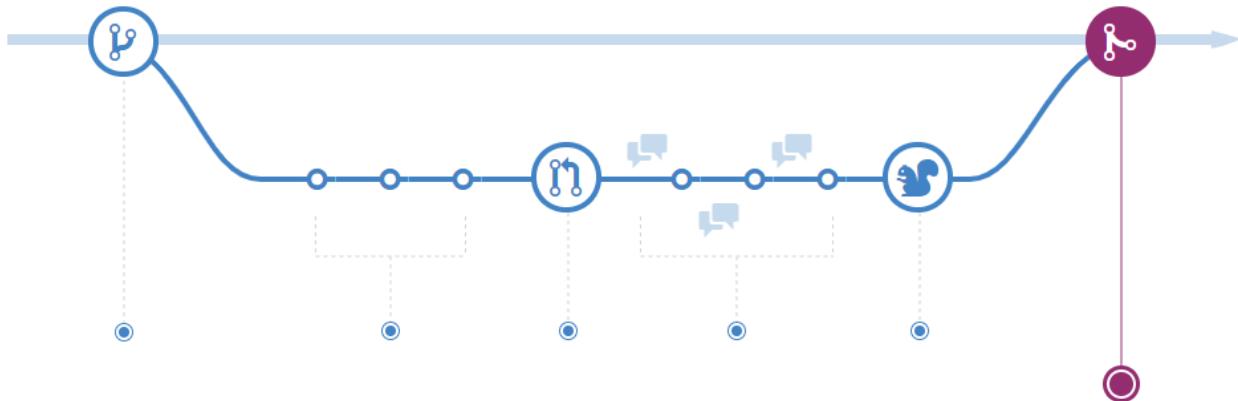
## Deploy



With Git, you can deploy from a branch for final testing in an environment before merging to the main.

Once your pull request has been reviewed and the branch passes your tests, you can deploy your changes to verify them. You can roll it back if your branch causes issues by deploying the existing main.

## Merge



Once your changes have been verified, it's time to merge your code into the main branch.

Once merged, Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.

You can associate issues with code by incorporating specific keywords into your Pull Request text. When your Pull Request is merged, the related issues can also close.

This workflow helps organize and track branches focused on business domain feature sets.

Other Git workflows, like the Git Forking Workflow and the Gitflow Workflow, are repo-focused and can use the Git Feature Branch Workflow to manage their branching models.

# Explore Git branch model for continuous delivery

Completed 100 XP

- 3 minutes

The purpose of writing code is to ship enhancements to your software.

A branching model that introduces too much process overhead doesn't help increase the speed of getting changes to customers—developing a branching model gives you enough padding not to ship poor-quality changes. But, at the same time doesn't introduce too many processes to slow you down.

The internet is full of branching strategies for Git; while there's no right or wrong, a perfect branching strategy works for your team!

You'll learn always to use the combination of feature branches and pull requests to have a ready-to-ship main branch.

## Getting ready

Let's cover the principles of what we suggest:

- The main branch:
  - The main branch is the only way to release anything to production.
  - The main branch should always be in a ready-to-release state.
  - Protect the main branch with branch policies.
  - Any changes to the main branch flow through pull requests only.
  - Tag all releases in the main branch with Git tags.
- The feature branch:
  - Use feature branches for all new features and bug fixes.

- Use feature flags to manage long-running feature branches.
- Changes from feature branches to the main only flow through pull requests.
- Name your feature to reflect its purpose.

List of branches:

```
CMDCopy
bugfix/description
features/feature-name
features/feature-area/feature-name
hotfix/description
users/username/description
users/username/workitem
```

- Pull requests:
  - Review and merge code with pull requests.
  - Automate what you inspect and validate as part of pull requests.
  - Tracks pull request completion duration and set goals to reduce the time it takes.

We'll be using the myWebApp created in the previous exercises. See [Describe working with Git locally](#).

In this recipe, we'll be using three trendy extensions from the marketplace:

- [Azure CLI](#): is a command-line interface for Azure.
- [Azure DevOps CLI](#): It's an extension of the Azure CLI for working with Azure DevOps and Azure DevOps Server designed to seamlessly integrate with Git, CI pipelines, and Agile tools. With the Azure DevOps CLI, you can contribute to your projects without leaving the command line. CLI runs on Windows, Linux, and Mac.
- [Git Pull Request Merge Conflict](#): This open-source extension created by Microsoft DevLabs allows you to review and resolve the pull request merge conflicts on the web. Any conflicts with the target branch must be resolved before a Git pull request can complete. With this extension, you can resolve these conflicts on the web as part of the pull request merge instead of doing the merge and resolving conflicts in a local clone.

The Azure DevOps CLI supports returning the query results in JSON, JSONC, YAML, YAMLC, table, TSV, and none. You can configure your preference by using the `configure` command.

# How to do it

## Important

You need to have the project created in the first Learning Path: [Describe working with Git locally.](#)

1. After you've cloned the main branch into a local repository, create a new feature branch, myFeature-1:

***myWebApp***

```
CMDCopy  
git checkout -b feature/myFeature-1
```

### **Output:**

*Switched to a new branch 'feature/myFeature-1'.*

2. Run the Git branch command to see all the branches. The branch showing up with an asterisk is the "currently-checked-out" branch:

***myWebApp***

```
CMDCopy  
git branch
```

### **Output:**

*feature/myFeature-1*

*main*

3. Make a change to the Program.cs file in the feature/myFeature-1 branch:

***myWebApp***

```
CMDCopy  
notepad Program.cs
```

4. Stage your changes and commit locally, then publish your branch to remote:

## ***myWebApp***

```
CMDCopy  
git status
```

### **Output:**

*On branch feature/myFeature-1 Changes not staged for commit: (use "git add <file>..." to update what will be committed) (use "git checkout -- <file>..." to discard changes in working directory) modified: Program.cs.*

## ***myWebApp***

```
CMDCopy  
git add .  
git commit -m "Feature 1 added to Program.cs"
```

### **Output:**

*[feature/myFeature-1 70f67b2] feature 1 added to program.cs 1 file changed,  
1 insertion(+).*

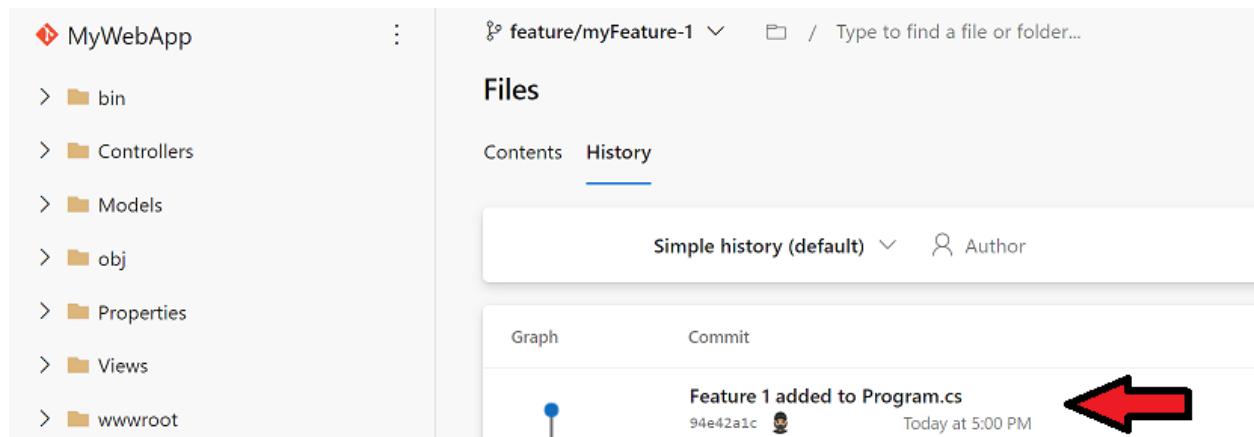
## ***myWebApp***

```
CMDCopy  
git push -u origin feature/myFeature-1
```

### **Output:**

*Delta compression using up to 8 threads. Compressing objects: 100% (3/3), done. Writing objects: 100% (3/3), 348 bytes | 348.00 KiB/s, done. Total 3 (delta 2), reused 0 (delta 0) remote: Analyzing objects... (3/3) (10 ms) remote: Storing packfile... done (44 ms) remote: Storing index... done (62 ms)  
To [https://dev.azure.com/Geeks/PartsUnlimited/\\_git/MyWebApp](https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp) \* [new branch] feature/myFeature-1 -> feature/myFeature-1 Branch  
feature/myFeature-1 set up to track remote branch feature/myFeature-1 from origin.*

The remote shows the history of the changes:



5. Configure Azure DevOps CLI for your organization and project.  
Replace **organization** and **project name**:

CMDCopy

```
az devops configure --defaults  
organization=https://dev.azure.com/organization project="project name"
```

6. Create a new pull request (using the Azure DevOps CLI) to review the changes in the feature-1 branch:

CMDCopy

```
az repos pr create --title "Review Feature-1 before merging to main" --  
work-items 38 39 `  
--description "#Merge feature-1 to main" `  
--source-branch feature/myFeature-1 --target-branch main `  
--repository myWebApp --open
```

Use the --open switch when raising the pull request to open it in a web browser after it has been created. The --deletesource-branch switch can be used to delete the branch after the pull request is complete. Also, consider using --auto-complete to complete automatically when all policies have passed, and the source branch can be merged into the target branch.

#### Note

For more information about **az repos pr create** parameter, see [Create a pull request to review and merge code](#).

The team jointly reviews the code changes and approves the pull request:

## Review Feature-1 before merging to main

Completed

!34



feature/myFeature-1 into main

Overview Files Updates Commits



completed this pull request Just now

Merged PR 34: Review Feature-1 before merging to main

0792185c



Just now

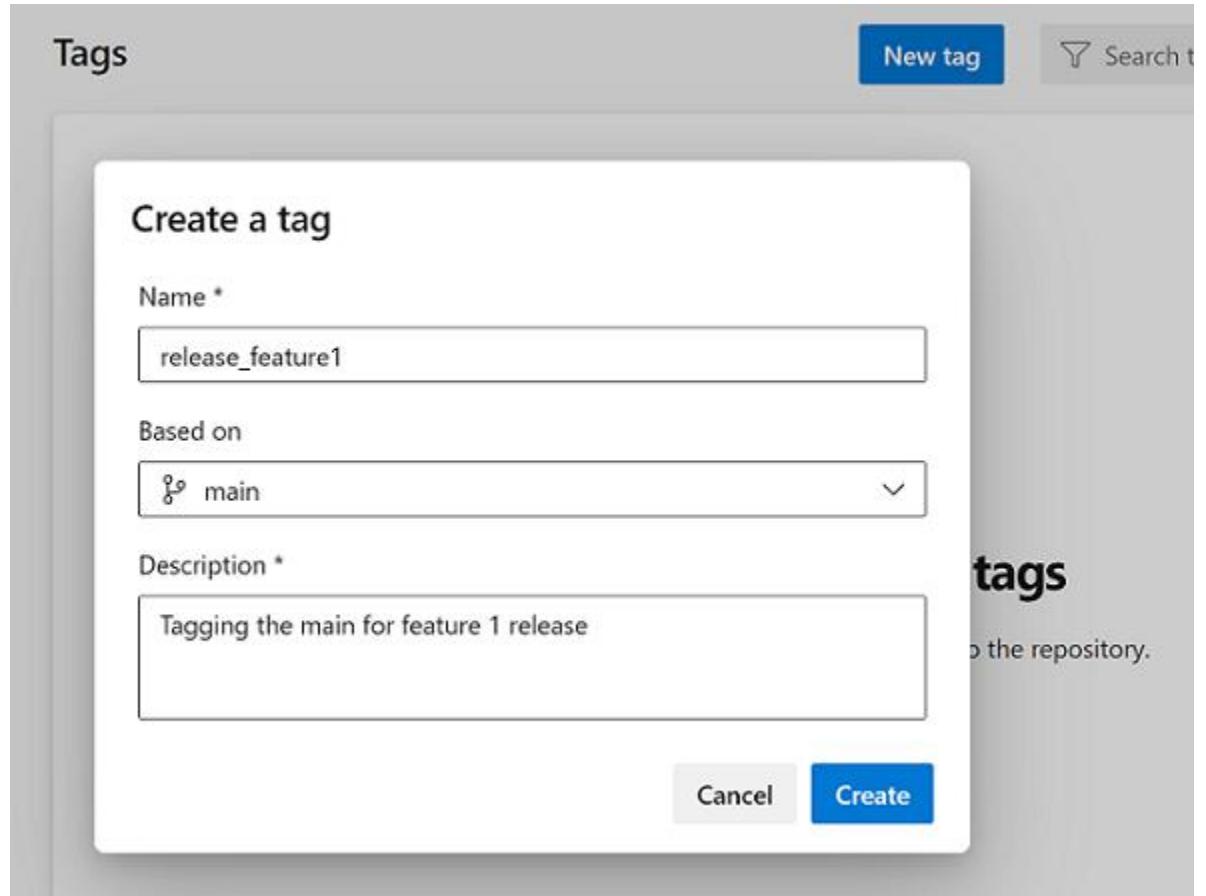
Show details



No merge conflicts

Last checked Just now

The main is ready to release. Team tags the main branch with the release number:



7. Start work on Feature 2. Create a branch on remote from the main branch and do the checkout locally:

### ***myWebApp***

CMDCopy

```
git push origin main:refs/heads/feature/myFeature-2
```

### **Output:**

*Total 0 (delta 0), reused 0 (delta 0)*

To [https://dev.azure.com/Geeks/PartsUnlimited/\\_git/MyWebApp](https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp) \* [new branch] origin/HEAD -> refs/heads/feature/myFeature-2.

### ***myWebApp***

CMDCopy

```
git checkout feature/myFeature-2
```

### **Output:**

*Switched to a new branch 'feature/myFeature-2' Branch feature/myFeature-2 set up to track remote branch feature/myFeature-2 from origin.*

8. Modify Program.cs by changing the same comment line in the code changed in feature-1.

**Copy**

```
public class Program
{
    // Editing the same line (file from feature-2 branch)
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();
}
```

9. Commit the changes locally, push them to the remote repository, and then raise a pull request to merge the changes from feature/myFeature-2 to the main branch:

**CMDCopy**

```
az repos pr create --title "Review Feature-2 before merging to main" --
work-items 40 42 ` 
--description "#Merge feature-2 to main" ` 
--source-branch feature/myFeature-2 --target-branch main ` 
--repository myWebApp --open
```

A critical bug is reported in production against the feature-1 release with the pull request in flight. To investigate the issue, you need to debug against the version of the code currently deployed in production. To investigate the issue, create a new fof branch using the release\_feature1 tag:

***myWebApp***

**CMDCopy**

```
git checkout -b fof/bug-1 release_feature1
```

**Output:**

*Switched to a new branch, 'fof/bug-1'.*

10. Modify Program.cs by changing the same line of code that was changed in the feature-1 release:

```
Copy
public class Program
{
    // Editing the same line (file from feature-FOF branch)
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();
}
```

11. Stage and commit the changes locally, then push changes to the remote repository:

### ***myWebApp***

```
CMDCopy
git add .
git commit -m "Adding FOF changes."
git push -u origin fof/bug-1
```

### **Output:**

To [https://dev.azure.com/Geeks/PartsUnlimited/\\_git/MyWebApp](https://dev.azure.com/Geeks/PartsUnlimited/_git/MyWebApp) \* [new branch] fof/bug-1 - fof/bug-1 Branch fof/bug-1 set up to track remote branch fof/bug-1 from origin.

12. Immediately after the changes have been rolled out to production, tag the fof/bug-1 branch with the release\_bug-1 tag, then raise a pull request to merge the changes from fof/bug-1 back into the main:

```
CMDCopy
az repos pr create --title "Review Bug-1 before merging to main" --work-items 100 ` 
--description "#Merge Bug-1 to main" ` 
--source-branch fof/Bug-1 --target-branch main ` 
--repository myWebApp --open
```

As part of the pull request, the branch is deleted. However, you can still reference the entire history using the tag.

With the critical bug fix out of the way, let's review the feature-2 pull request.

The branches page makes it clear that the feature/myFeature-2 branch is one change ahead of the main and two changes behind the main:

A screenshot of a GitHub branches page. The page title is "Branches". At the top right is a "New branch" button. Below the title are filters: "Mine", "All", and "Stale". A search bar with the placeholder "Search branch name" is also present. The main table has columns: "Branch", "Commit", "Author", "Author...", "Behind | Ahead", "Status", and "Pull Re...". There are two entries under the "feature" folder: "myFeature-2" and "main". "myFeature-2" has a commit hash of "e0c8aa03", author "Lu...", date "7m ago", and status "2|1" (two commits ahead, one behind). "main" has a commit hash of "8792185c", author "Lu...", date "23m a...", and status "1|1" (one commit ahead, one behind). A red arrow points to the "2|1" status of the "myFeature-2" row.

| Branch      | Commit   | Author | Author... | Behind   Ahead | Status | Pull Re... |
|-------------|----------|--------|-----------|----------------|--------|------------|
| feature     |          |        |           |                |        |            |
| myFeature-2 | e0c8aa03 | Lu...  | 7m ago    | 2 1            | 11 35  | ★          |
| main        | 8792185c | Lu...  | 23m a...  | 1 1            | 11 35  | ★          |

If you tried to approve the pull request, you'd see an error message informing you of a merge conflict:

A screenshot of a GitHub pull request review for "Review Feature-2 before merging to master". The pull request is from "fof/bug-1" into "main". It shows 136 changes and 1 merge conflict. The "Overview" tab is selected. The "Files" tab shows a conflict in "C# Program.cs" with the status "Edited in both". The "Commits" tab is also visible. In the "Description" section, the merge message is "Merge feature-2 to master".

13. The Git Pull Request Merge Conflict resolution extension makes it possible to resolve merge conflicts right in the browser. Navigate to the conflicts tab and click on Program.cs to resolve the merge conflicts:

```

Review Feature-2before merging to master
Active 136 fof/bug-1 into main
Overview Files Updates Commits
All Changes Filter Program.cs +1 /Program.cs
C# Program.cs
1 var builder = WebApplication.CreateBuilder(args);
2
3 // Add services to the container.
4 builder.Services.AddControllersWithViews();
5
6 var app = builder.Build();
7
8 // Configure the HTTP request pipeline.
9 if (app.Environment.IsDevelopment())
10 {
11     app.UseExceptionHandler("/Home/Error");
12     // The default HSTS value is 30 days. You may want to c
13     app.UseHsts();
14 }
15
16 app.UseHttpsRedirection();
17 app.UseStaticFiles();
18
19 app.UseRouting();
20
21 app.UseAuthorization();
22
23 app.MapControllerRoute(
24     name: "default",
25     pattern: "{controller=Home}/{action=Index}/{id?}");
26
27 app.Run();
28
29
1 var builder = WebApplication.CreateBuilder(args);
2
3 // Add services to the container.
4 builder.Services.AddControllersWithViews();
5
6 var app = builder.Build();
7
8 // Configure the HTTP request pipeline.
9 if (app.Environment.IsDevelopment())
10 {
11     app.UseExceptionHandler("/Home/Error");
12     // The default HSTS value is 30 days. You may want to c
13     app.UseHsts();
14 }
15
16 app.UseHttpsRedirection();
17 app.UseStaticFiles();
18
19 app.UseRouting();
20
21 // Authorization (file from feature-FOF branch)
22 app.UseAuthorization();
23
24 app.MapControllerRoute(
25     name: "default",
26     pattern: "{controller=Home}/{action=Index}/{id?}");
27
28 app.Run();
29

```

The user interface allows you to take the source, target, add custom changes, review, and submit the merge. With the changes merged, the pull request is completed.

## How it works

We learned how the Git branching model gives you the flexibility to work on features in parallel by creating a branch for each feature.

The pull request workflow allows you to review code changes using the branch policies.

Git tags are a great way to record milestones, such as the version of code released; tags give you a way to create branches from tags.

We created a branch from a previous release tag to fix a critical bug in production.

The branches view in the web portal makes it easy to identify branches ahead of the main. Also, it forces a merge conflict if any ongoing pull requests try to merge to the main without resolving the merge conflicts.

A lean branching model allows you to create short-lived branches and push quality changes to production faster.

# Explore GitHub flow

Completed 100 XP

- 3 minutes

GitHub is the best tool to enable collaboration in your projects. GitHub flow is a branch-based workflow suggested for GitHub.

## Note

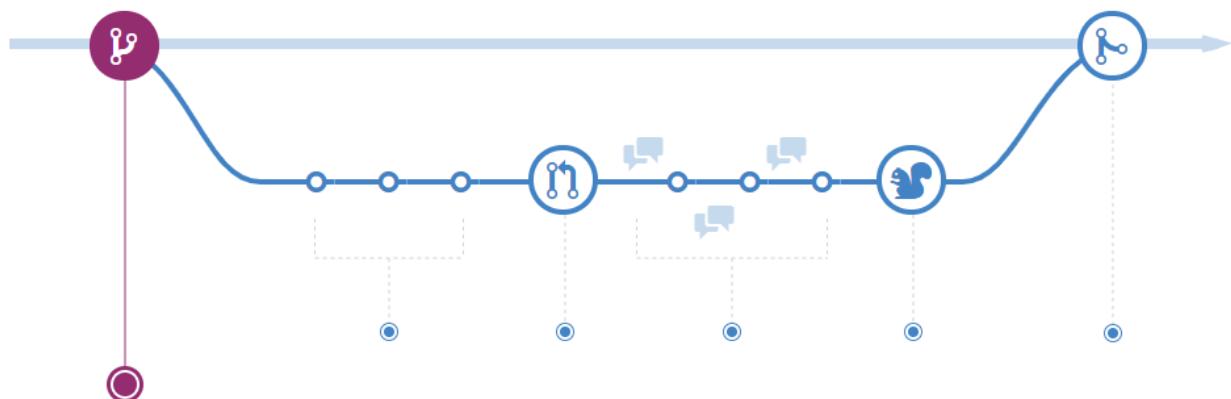
To implement GitHub flow, you'll need a GitHub account and a repository. See "[Signing up for GitHub](#)" and "[Create a repo](#)."

## Tip

You can complete all steps of GitHub flow through the GitHub web interface, command line, [GitHub CLI](#), or [GitHub Desktop](#).

The first step is to create a branch in your repository to work without affecting the default branch, and you give collaborators a chance to review your work.

For more information, see "[Creating and deleting branches within your repository](#)."



Make any desired changes to the repository. If you make a mistake, you can revert or push extra changes to fix it.

Commit and push your changes to your branch to back up your work to remote storage, giving each commit a descriptive message. Each commit should contain an isolated, complete change making it easy to revert if you take a different approach.

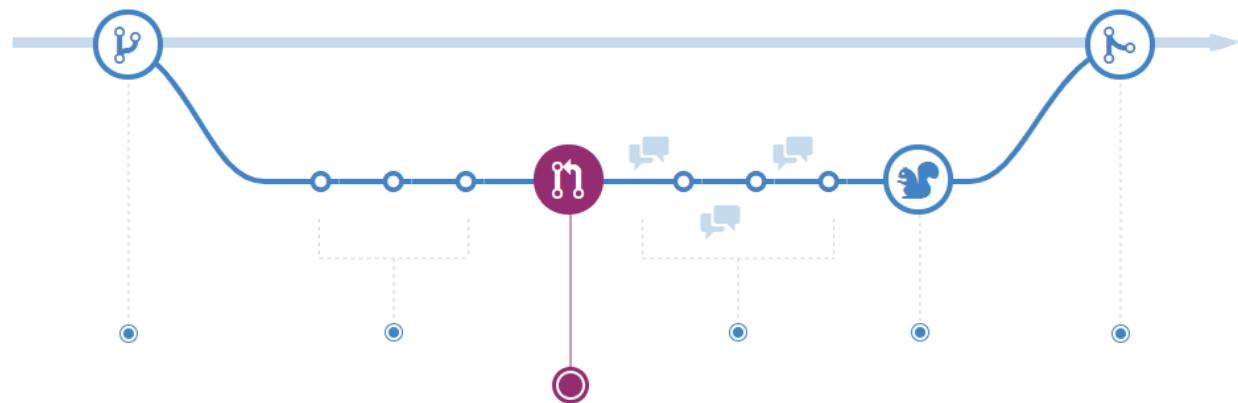
Anyone collaborating with your project can see your work, answer questions, and make suggestions or contributions. Continue to create, commit, and push changes to your branch until you're ready to ask for feedback.

### Tip

You can make a separate branch for each change to make it easy for reviewers to give feedback or for you to understand the differences.

Once you're ready, you can create a pull request to ask collaborators for feedback on your changes. See "[Creating a pull request](#)."

Pull request review is one of the most valuable features of collaboration. You can require approval from your peers and team before merging changes. Also, you can mark it as a draft in case you want early feedback or advice before you complete your changes.



Describe the pull request as much as possible with the suggested changes and what problem you're resolving. You can add images, links, related issues, or any information to document your change and help reviewers understand the PR without opening each file. See "[Basic writing and formatting syntax](#)" and "[Linking a pull request to an issue](#)."

John Doe commented on Mar 4

Collaborator

- Overall - Minor typos and word capitalizations corrected.
- Lab numbering and ordering to follow Skillpipe and Slides order.

## Related Issue

---

#213

## Checklist

---

Mark completed with "x" between brackets, "[x]", or checking the box once the PR is created:

- Has related GitHub Issue [Create Issue](#)
- Tested it
- Read the PR collaboration guide [Collaboration Guide](#)

Changes proposed in this pull request:

- Lab numbering and ordering to follow Skillpipe and Slides order.

Another way to improve PR quality and documentation and explicitly point something out to the reviewers is to use the comment session area. Also, you can @mention or request a review from specific people or teams.

Write Preview H B I ≡ ↔ ∅ ☰ ☒ @ ✉ ↶

Leave a comment

---

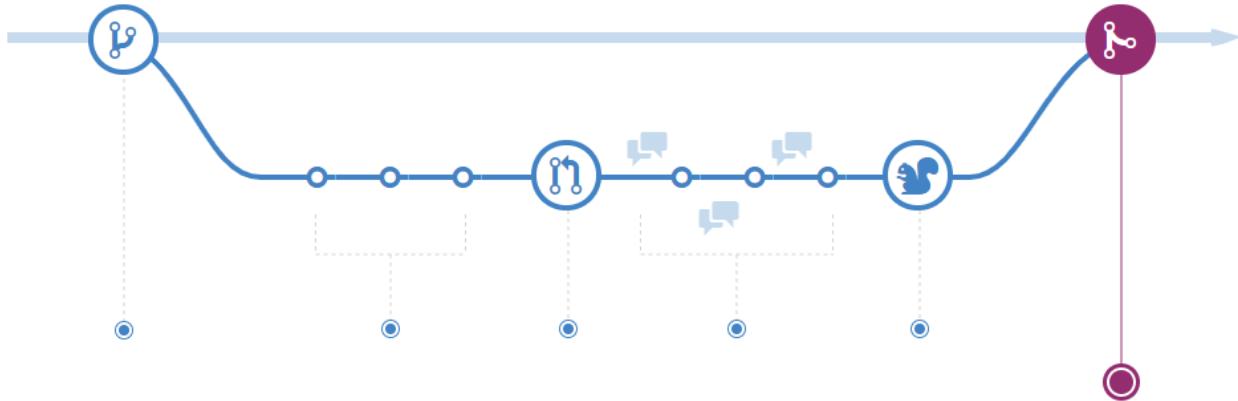
Attach files by dragging & dropping, selecting or pasting them.

M4

There are other Pull Requests configurations, such as automatically requesting a review from specific teams or users when a pull request is created or checks to run on pull requests. For more information, see "[About status checks](#)" and "[About protected branches](#)".

After the reviewers' comments and checks validation, the changes should be ready to be merged, and they can approve the Pull Request. See [Merging a pull request](#).

If you have any conflicts, GitHub will inform you to resolve them. "[Addressing merge conflicts](#)."



After a successful pull request merges, there's no need for the remote branch to stay there. You can delete your branch to prevent others from accidentally using old branches. For more information, see "[Deleting and restoring branches in a pull request](#)."

### Note

GitHub keeps the commit and merges history if you need to restore or revert your pull request.

## Explore fork workflow

Completed 100 XP

- 3 minutes

The forking workflow is fundamentally different than other popular Git workflows.

Instead of using a single server-side repository to act as the "central" codebase, it gives every developer their server-side repository.

It means that each contributor has two Git repositories:

- A private local.
- A public server-side.

The forking workflow is most often seen in public open-source projects.

The main advantage of the forking workflow is that contributions can be integrated without the need for everybody to push to a single central repository.

Developers push to their server-side repositories, and only the project maintainer can push to the official repository.

It allows the maintainer to accept commits from any developer without giving them written access to the official codebase.

The forking workflow typically will be intended for merging into the original project maintainer's repository.

The result is a distributed workflow that provides you a flexible way for large, organic teams (including untrusted third parties) to collaborate securely.

This also makes it an ideal workflow for open-source projects.

## How it works

As in the other Git workflows, the forking workflow begins with an official public repository stored on a server.

But when a new developer wants to start working on the project, they don't directly clone the official repository.

Instead, they fork the official repository to create a copy of it on the server.

This new copy serves as their personal public repository—no other developers can push to it, but they can pull changes from it (we'll see why this is necessary in a moment).

After they've created their server-side copy, the developer does a git clone to get a copy of it onto their local machine.

It serves as their private development environment, just like in the other workflows.

When they're ready to publish a local commit, they push the commit to their public repository—not the official one.

Then, they file a pull request with the main repository, which lets the project maintainer know that an update is ready to be integrated.

The pull request also serves as a convenient discussion thread if there are issues with the contributed code.

The following is a step-by-step example of this workflow:

- A developer 'forks' an 'official' server-side repository. It creates their server-side copy.
- The new server-side copy is cloned to their local system.
- A Git remote path for the 'official' repository is added to the local clone.
- A new local feature branch is created.
- The developer makes changes to the new branch.
- New commits are created for the changes.
- The branch gets pushed to the developer's server-side copy.
- The developer opens a pull request from the new branch to the 'official' repository.
- The pull request gets approved for merge and is merged into the original server-side repository.

To integrate the feature into the official codebase:

- The maintainer pulls the contributor's changes into their local repository.
- Checks to make sure it doesn't break the project.
- Merges it into their local main branch.
- Pushes the main branch to the official repository on the server.

The contribution is now part of the project, and other developers should pull from the official repository to synchronize their local repositories.

It's essential to understand that the notion of an "official" repository in the forking workflow is merely a convention.

The only thing that makes the official repository, so official is that it's the repository of the project maintainer.

## Forking vs. cloning

It's essential to note that "forked" repositories and "forking" aren't special operations.

Forked repositories are created using the standard `git clone` command. Forked repositories are generally "server-side clones" managed and hosted by a Git service provider such as Azure Repos.

There's no unique Git command to create forked repositories.

A clone operation is essentially a copy of a repository and its history.

# Version Control with Git in Azure Repos

Completed 100 XP

- 60 minutes

**Estimated time:** 60 minutes.

**Lab files:** none.

## Scenario

Azure DevOps supports two types of version control, Git and Team Foundation Version Control (TFVC). Here's a quick overview of the two version control systems:

- **Team Foundation Version Control (TFVC):** TFVC is a centralized version control system. Typically, team members have only one version of each file on their dev machines. Historical data is maintained only on the server. Branches are path-based and created on the server.
- **Git:** Git is a distributed version control system. Git repositories can live locally (on a developer's machine). Each developer has a copy of the source repository on their dev machine. Developers can commit each set of changes on their dev machine, perform version control operations such as history, and compare without a network connection.

Git is the default version control provider for new projects. You should use Git for version control in your projects unless you need centralized version control features in TFVC.

In this lab, you'll learn to establish a local Git repository, which can easily be synchronized with a centralized Git repository in Azure DevOps. In addition, you'll learn about Git branching and merging support. You'll use Visual Studio Code, but the same processes apply to using any Git-compatible client.

## Objectives

After completing this lab, you'll be able to:

- Clone an existing repository.
- Save work with commits.
- Review the history of changes.
- Work with branches by using Visual Studio Code.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- If you don't have Git **2.29.2** or later installed, start a web browser, navigate to the [Git for Windows download page](#), and install it.
- If you don't have Visual Studio Code installed yet, navigate to the [Visual Studio Code download page](#) from the web browser window, download it, and install it.
- If you don't have the Visual Studio C# extension installed yet, navigate to the [C# extension installation page](#) in the web browser window and install it.

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Clone an existing repository.
- Exercise 2: Save work with commits.
- Exercise 3: Review history.
- Exercise 4: Work with branches.

# Collaborate with pull requests in Azure Repos

## Collaborate with pull requests

Completed 100 XP

- 3 minutes

Pull requests let you tell others about changes you've pushed to a GitHub repository.

Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.

Pull requests are commonly used by teams and organizations collaborating using the Shared Repository Model.

Everyone shares a single repository, and topic branches are used to develop features and isolate changes.

Many open-source projects on GitHub use pull requests to manage changes from contributors.

They help provide a way to notify project maintainers about changes one has made.

Also, start code review and general discussion about a set of changes before being merged into the main branch.

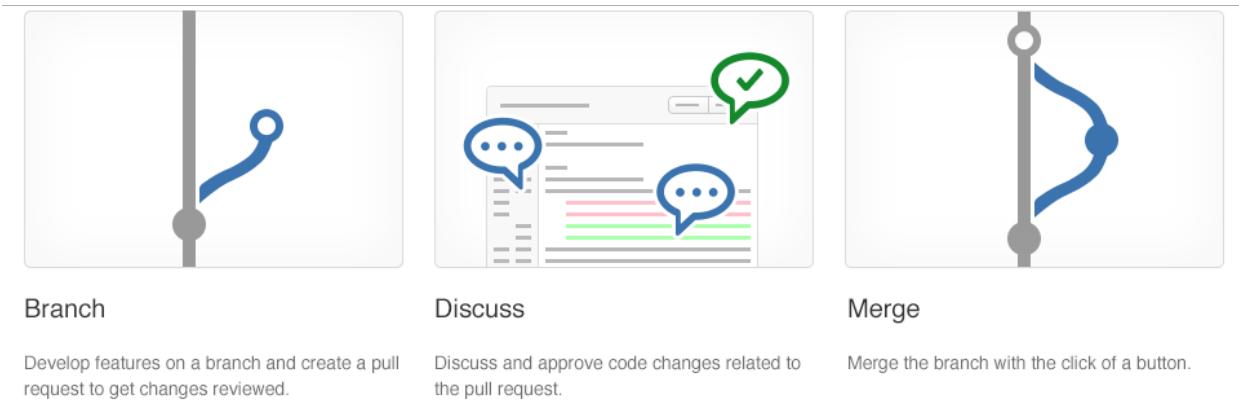
Pull requests combine the review and merge of your code into a single collaborative process.

Once you're done fixing a bug or new feature in a branch, create a new pull request.

Add the team members to the pull request so they can review and vote on your changes.

Use pull requests to review works in progress and get early feedback on changes.

There's no commitment to merge the changes as the owner can abandon the pull request at any time.



## Get your code reviewed

The code review done in a pull request isn't just to find bugs—that's what your tests are concerning.

A good code review catches less obvious problems that could lead to costly issues later.

Code reviews help protect your team from bad merges and broken builds that sap your team's productivity.

The review catches these problems before the merge, protecting your essential branches from unwanted changes.

Cross-pollinate expertise and spread problem-solving strategies by using a wide range of reviewers in your code reviews.

Diffusing skills and knowledge makes your team more robust and more resilient.

## Give great feedback

High-quality reviews start with high-quality feedback. The keys to great feedback in a pull request are:

- Have the right people review the pull request.
- Make sure that reviewers know what the code does.
- Give actionable, constructive feedback.
- Reply to comments promptly.

When assigning reviewers to your pull request, make sure you select the right set of reviewers.

You want reviewers who know how your code works and try to include developers working in other areas to share their ideas.

Also, who can provide a clear description of your changes and build your code that has your fix or feature running in it.

Reviewers should try to provide feedback on changes they disagree with. Identify the issue and give a specific suggestion on what you would do differently.

This feedback has clear intent and is easy for the owner of the pull request to understand.

The pull request owner should reply to the comments, accept the suggestion, or explain why the suggested change isn't ideal.

Sometimes a suggestion is good, but the changes are outside the scope of the pull request.

Take these suggestions and create new work items and feature branches separate from the pull request to make those changes.

## Protect branches with policies

There are a few critical branches in your repo that the team relies on always in suitable shapes, such as your main branch.

Require pull requests to make any changes on these branches. Developers pushing changes directly to the protected branches will have their pushes rejected.

Add more conditions to your pull requests to enforce a higher level of code quality in your key branches.

A clean build of the merged code and approval from multiple reviewers are extra requirements you can set to protect your key branches.

# Exercise - Azure Repos collaborating with pull requests

Completed 100 XP

- 6 minutes

Code issues that are found sooner are both easier and cheaper to fix. So development teams strive to push code quality checks as far left into the development process as possible.

As the name suggests, branch policies give you a set of out-of-the-box policies that can be applied to the branches on the server.

Any changes being pushed to the server branches need to follow these policies before the changes can be accepted.

Policies are a great way to enforce your team's code quality and change-management standards. In this recipe, you'll learn how to configure branch policies on your main branch.

## Getting ready

The out-of-the-box branch policies include several policies, such as build validation and enforcing a merge strategy. We'll only focus on the branch policies needed to set up a code-review workflow in this recipe.

## How to do it

1. Open the branches view for the myWebApp Git repository in the parts-unlimited team portal. Select the main branch, and from the pull-down, context menu choose Branch policies:

The screenshot shows the Azure DevOps interface for managing branches. On the left, there's a sidebar with various options like Overview, Boards, Repos, and Branches. The 'Branches' option is selected and highlighted with a red box. The main area is titled 'Branches' and shows a table with columns for Branch, Author, and Status. A specific row for the 'main' branch is selected, showing 879 commits. On the right, there's a context menu with several options, one of which, 'Branch policies', is also highlighted with a red box.

2. In the policies view, It presents out-of-the-box policies. Set the minimum number of reviewers to 1:

This screenshot shows the 'Branch Policies' configuration page. At the top, it says 'Branch Policies' and includes a note: 'Note: If any required policy is enabled, this branch cannot be deleted and changes must be made via pull request.' Below this, there's a toggle switch labeled 'On'. Underneath the switch, there's a section for 'Require a minimum number of reviewers' with a description: 'Require approval from a specified number of reviewers on pull requests.' A text input field contains the value '1'. Below this, there are four optional checkboxes:

- Allow requestors to approve their own changes
- Prohibit the most recent pusher from approving their own changes
- Allow completion even if some reviewers vote to wait or reject
- When new changes are pushed:

The Allow requestors to approve their own changes option allows the submitter to self-approve their changes.

It's OK for mature teams, where branch policies are used as a reminder for the checks that need to be performed by the individual.

3. Use the review policy with the comment-resolution policy. It allows you to enforce that the code review comments are resolved before the changes are accepted. The requester can take the feedback from the comment and create a new work item and resolve the changes. It at least guarantees that code review comments aren't lost with the acceptance of the code into the main branch:

 On

**Check for comment resolution**  
Check to see that all comments have been resolved on pull requests.

**Required**  
Block pull requests from being completed while any comments are active.

**Optional**  
Warn if any comments are active, but allow pull requests to be completed.

4. A requirement instigates a code change in the team project. If the work item triggered the work isn't linked to the change, it becomes hard to understand why it was made over time. It's especially useful when reviewing the history of changes. Configure the Check for linked work items policy to block changes that don't have a work item linked to them:

 On

**Check for linked work items**  
Encourage traceability by checking for linked work items on pull requests.

**Required**  
Block pull requests from being completed unless they have at least one linked work item.

**Optional**  
Warn if there are no linked work items, but allow pull requests to be completed.

5. Select the option to automatically include reviewers when a pull request is raised automatically. You can map which reviewers are added based on the area of the code being changed:

## Add new reviewer policy

Reviewers

Add people

Policy requirement

Optional

Required

For pull request affecting these folders

ⓘ

Leave blank to include the specified reviewers on all pull requests

Completion options

Allow requestors to approve their own changes

Activity feed message

ⓘ

## How it works

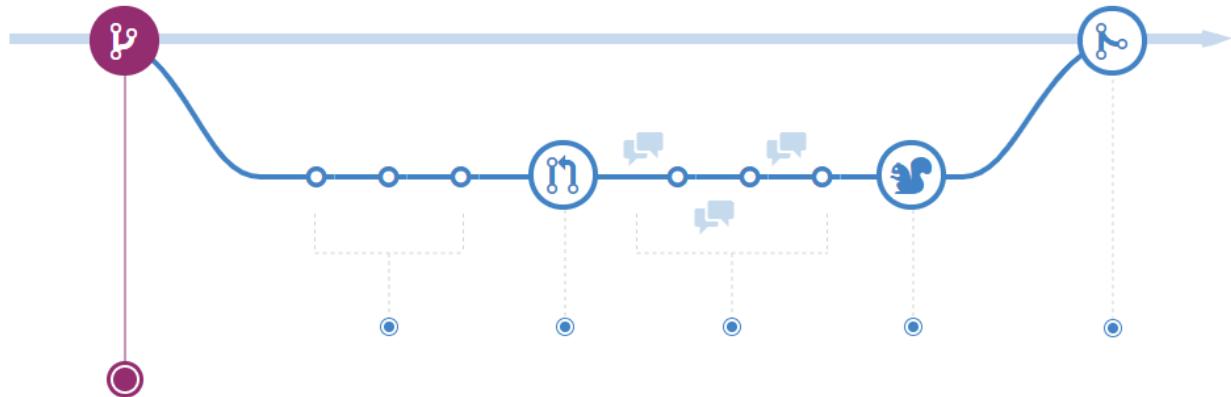
With the branch policies in place, the main branch is now fully protected.

The only way to push changes to the main branch is by first making the changes in another branch and then raising a pull request to trigger the change-acceptance workflow.

Choose to create a new branch from one of the existing user stories in the work item hub.

By creating a new branch from a work item, that work item automatically gets linked to the branch.

You can optionally include more than one work item with a branch as part of the create workflow:



Prefix in the name when creating the branch to make a folder for the branch to go in.

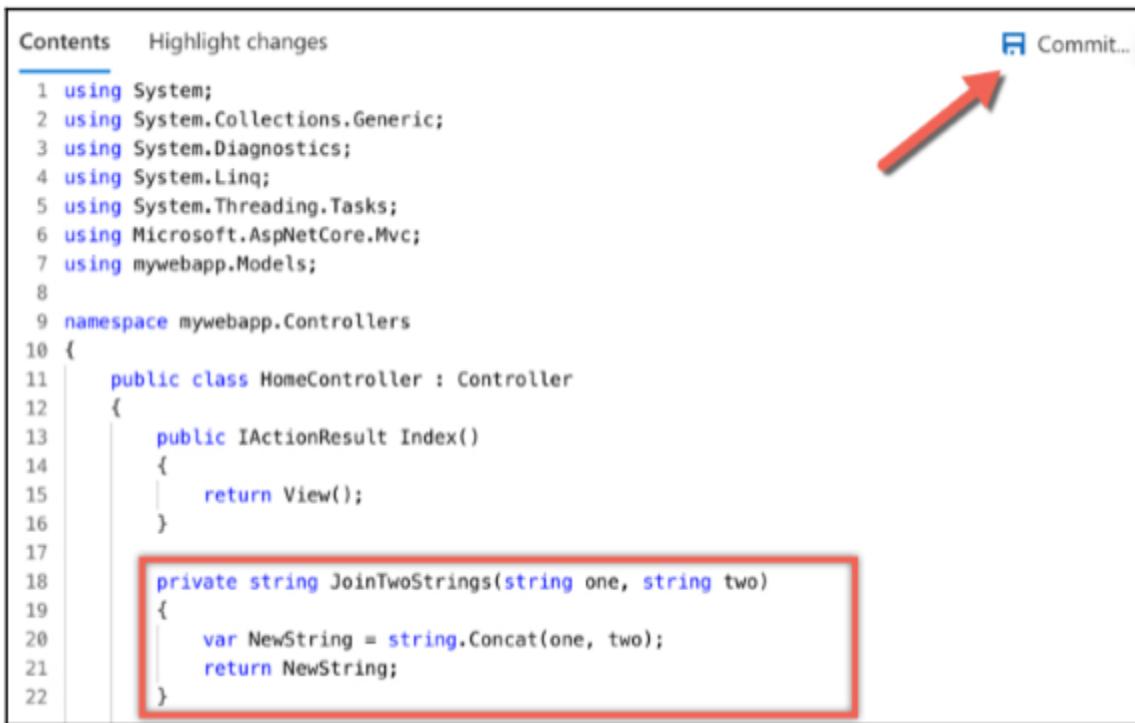
In the preceding example, the branch will go in the folder. It is a great way to organize branches in busy environments.

With the newly created branch selected in the web portal, edit the `HomeController.cs` file to include the following code snippet and commit the changes to the branch.

In the image below, you'll see that you can directly commit the changes after editing the file by clicking the commit button.

The file path control in the team portal supports search.

Start typing the file path to see all files in your Git repository under that directory, starting with these letters showing up in the file path search results dropdown.



The screenshot shows a code editor interface within the Azure DevOps web portal. The code is written in C# and defines a HomeController class with an Index method and a private helper method JoinTwoStrings. A red box highlights the JoinTwoStrings method, and a red arrow points from the bottom right towards the 'Commit...' button in the top right corner.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Diagnostics;
4 using System.Linq;
5 using System.Threading.Tasks;
6 using Microsoft.AspNetCore.Mvc;
7 using mywebapp.Models;
8
9 namespace mywebapp.Controllers
10 {
11     public class HomeController : Controller
12     {
13         public IActionResult Index()
14         {
15             return View();
16         }
17
18         private string JoinTwoStrings(string one, string two)
19         {
20             var NewString = string.Concat(one, two);
21             return NewString;
22         }
23     }
24 }
```

The code editor in the web portal has several new features in Azure DevOps Server, such as support for bracket matching and toggle white space.

You can load the command palette by pressing it. Among many other new options, you can now toggle the file using a file mini-map, collapse, and expand, and other standard operations.

To push these changes from the new branch into the main branch, create a pull request from the pull request view.

Select the new branch as the source and the main as the target branch.

The new pull request form supports markdown, so you can add the description using the markdown syntax.

The description window also supports @ mentions and # to link work items:

Description

# Updated HomeController.cs  
+ Added a new method for concatenating two strings

Markdown supported.

Aa **B** *I*  $\mathcal{D}$   $\leftrightarrow$   $\equiv$   $\equiv$   $\equiv$  @ #

## Updated HomeController.cs

- Added a new method for concatenating two strings

Reviewers

[PartsUnlimited] \ PartsUnlimited Team  Search users and groups to add as reviewers

Work Items

Search work items by ID or title

  38 Retrieve the user profile for personalization settings

The pull request is created; the overview page summarizes the changes and the status of the policies.

The Files tab shows you a list of changes and the difference between the previous and the current versions.

Any updates pushed to the code files will show up in the Updates tab, and a list of all the commits is shown under the Commits tab:

The screenshot shows the 'Pull Requests' tab for a repository named 'MyWebApp'. There is one active pull request titled 'Updated HomeController.cs' with the commit message 'sprint1/38-userProfilePersonalizationSettings into master'. The pull request has 0 reviews and 1 update. The 'Description' section includes a bullet point: 'Added a new method for concatenating two strings'. The 'Policies' section shows 'Required' status: 0 of 1 reviewers approved (red X), Work items linked (green checkmark), and All comments resolved (green checkmark). The 'Work Items' section lists a single item: '3B Retrieve the user profile for personali...'. The 'Reviewers' section shows 'PartsUnlimited Team' assigned as a reviewer.

Open the Files tab: this view supports code comments at the line level, file level, and overall.

The comments support both @ for mentions and # to link work items, and the text supports markdown syntax:

The code comments are persisted in the pull request workflow; the code comments support multiple iterations of reviews and work well with nested responses.

The reviewer policy allows for a code review workflow as part of the change acceptance.

It's an excellent way for the team to collaborate on any code changes pushed into the main branch.

When the required number of reviewers approves the pull request, it can be completed.

You can also mark the pull request to autocomplete after your review. It autocompletes the pull requests once all the policies have been successfully compiled.

## There's more

Have you ever been in a state where a branch has been accidentally deleted? It can't be easy to figure out what happened.

Azure DevOps Server now supports searching for deleted branches. It helps you understand who deleted it and when. The interface also allows you to recreate the branch.

Deleted branches are only shown if you search for them by their exact name to cut out the noise from the search results.

To search for a deleted branch, enter the full branch name into the branch search box. It will return any existing branches that match that text.

You'll also see an option to search for an exact match in the list of deleted branches.

If a match is found, you'll see who deleted it and when. You can also restore the branch. Restoring the branch will re-create it at the commit to which it last pointed.

However, it won't restore policies and permissions.

## Examine GitHub mobile for pull request approvals

Completed 100 XP

- 1 minute



4.8 ★★★★★

271 Ratings

In-App Purchases

...

#2

4+

Developer Tools

Age

The screenshot shows a GitHub mobile application interface. At the top, there's a header with the GitHub logo and the text "Projects, ideas, & code to go". Below the header, there's a blue button labeled "GET" and a "In-App Purchases" link. To the right of the "GET" button is a three-dot menu icon.

The main content area displays a pull request titled "Upgrade Electron from v5 to v7". The pull request has a status of "Merged". It shows 28 files changed, with 599 additions and 274 deletions. A "Review" button is visible next to the file change details.

Below the pull request, there's a section for "Commits" with a count of 40. A user named "kuychaco" is listed as having edited the pull request 8 months ago.

On the right side of the screen, there's a detailed view of the diff for the "app/.npmrc" file, showing changes from version 5.0.6 to 7.1.8. The diff highlights added lines in green and deleted lines in red.

Using a mobile app in combination with Git is a convenient option, particularly when urgent pull request approvals are required.

- The app can render markdown, images, and PDF files directly on the mobile device.
- Pull requests can be managed within the app, along with marking files as viewed, collapsing files.
- Comments can be added.
- Emoji short codes are rendered.

# Identify technical debt

## Examine code quality

Completed 100 XP

- 2 minutes

The quality of code shouldn't be measured subjectively. A developer-writing code would rate the quality of their code high, but that isn't a great way to measure code quality. Different teams may use different definitions based on context.

Code that is considered high quality may mean one thing for an automotive developer. And it may mean another for a web application developer.

The quality of the code is essential, as it impacts the overall software quality.

A study on "Software Defect Origins and Removal Methods" found that individual programmers are less than 50% efficient at finding bugs in their software. And most forms of testing are only 35% efficient. It makes it difficult to determine quality.

There are five key traits to measure for higher quality.

## Reliability

Reliability measures the probability that a system will run without failure over a specific period of operation. It relates to the number of defects and availability of the software. Several defects can be measured by running a static analysis tool.

Software availability can be measured using the mean time between failures (MTBF).

Low defect counts are crucial for developing a reliable codebase.

## Maintainability

Maintainability measures how easily software can be maintained. It relates to the codebase's size, consistency, structure, and complexity. And ensuring maintainable source code relies on several factors, such as testability and understandability.

You can't use a single metric to ensure maintainability.

Some metrics you may consider to improve maintainability are the number of stylistic warnings and Halstead complexity measures.

Both automation and human reviewers are essential for developing maintainable codebases.

## Testability

Testability measures how well the software supports testing efforts. It relies on how well you can control, observe, isolate, and automate testing, among other factors.

Testability can be measured based on how many test cases you need to find potential faults in the system.

The size and complexity of the software can impact testability.

So, applying methods at the code level—such as cyclomatic complexity—can help you improve the testability of the component.

## Portability

Portability measures how usable the same software is in different environments. It relates to platform independence.

There isn't a specific measure of portability. But there are several ways you can ensure portable code.

It's essential to regularly test code on different platforms rather than waiting until the end of development.

It's also good to set your compiler warning levels as high as possible and use at least two compilers.

Enforcing a coding standard also helps with portability.

## Reusability

Reusability measures whether existing assets—such as code—can be used again.

Assets are more easily reused if they have modularity or loose coupling characteristics.

The number of interdependencies can measure reusability.

Running a static analyzer can help you identify these interdependencies.

## Examine complexity and quality metrics

Completed 100 XP

- 2 minutes

While there are various quality metrics, a few of the most important ones are listed here.

Complexity metrics can help in measuring quality. Cyclomatic complexity measures the number of linearly independent paths through a program's source code. Another way to understand quality is through calculating Halstead complexity measures.

This measure:

- Program vocabulary.
- Program length.
- Calculated program length.
- Volume.
- Difficulty.
- Effort.

Code analysis tools can be used to check for security, performance, interoperability, language usage, and globalization and should be part of every developer's toolbox and software build process.

Regularly running a static code analysis tool and reading its output is a great way to improve as a developer because the things caught by the software rules can often teach you something.

## Common-quality-related metrics

One of the promises of DevOps is to deliver software both faster and with higher quality. Previously, these two metrics have been almost opposites. The more quickly you went, the lower the quality. The higher the quality, the longer it took. But DevOps

processes can help you find problems earlier, which usually means that they take less time to fix.

We've previously talked about some general project metrics and KPIs. The following is a list of metrics that directly relate to the quality of the code being produced and the build and deployment processes.

- **Failed builds percentage** - Overall, what percentage of builds are failing?
- **Failed deployments percentage** - Overall, what percentage of deployments are failing?
- **Ticket volume** - What is the overall volume of customer or bug tickets?
- **Bug bounce percentage** - What percentage of customer or bug tickets are reopened?
- **Unplanned work percentage** - What percentage of the overall work is unplanned?

## Introduction to technical debt

Completed 100 XP

- 4 minutes

**Technical debt** is a term that describes the future cost that will be incurred by choosing an easy solution today instead of using better practices because they would take longer to complete.

The term-technical debt was chosen for its comparison to financial debt. It's common for people in financial debt to make decisions that seem appropriate or the only option at the time, but in so doing, interest increases.

The more interest that accumulates, the harder it is for them in the future and the more minor options available to them later. With financial debt, soon, interest increases on interest, creating a snowball effect. Similarly, technical debt can build up to the point where developers spend almost all their time sorting out problems and doing rework, either planned or unplanned, rather than adding value.

So, how does it happen?

The most common excuse is tight deadlines. When developers are forced to create code quickly, they'll often take shortcuts. For example, instead of refactoring a method to include new functionality, let us copy to create a new version. Then I only test my new

code and can avoid the level of testing required if I change the original method because other parts of the code use it.

Now I have two copies of the same code that I need to modify in the future instead of one, and I run the risk of the logic diverging. There are many causes. For example, there might be a lack of technical skills and maturity among the developers or no clear product ownership or direction.

The organization might not have coding standards at all. So, the developers didn't even know what they should be producing. The developers might not have precise requirements to target. Well, they might be subject to last-minute requirement changes.

Necessary-refactoring work might be delayed. There might not be any code quality testing, manual or automated. In the end, it just makes it harder and harder to deliver value to customers in a reasonable time frame and at a reasonable cost.

Technical debt is one of the main reasons that projects fail to meet their deadlines.

Over time, it increases in much the same way that monetary debt does. Common sources of technical debt are:

- Lack of coding style and standards.
- Lack of or poor design of unit test cases.
- Ignoring or not understanding object oriented design principles.
- Monolithic classes and code libraries.
- Poorly envisioned the use of technology, architecture, and approach. (Forgetting that all system attributes, affecting maintenance, user experience, scalability, and others, need to be considered).
- Over-engineering code (adding or creating code that isn't required, adding custom code when existing libraries are sufficient, or creating layers or components that aren't needed).
- Insufficient comments and documentation.
- Not writing self-documenting code (including class, method, and variable names that are descriptive or indicate intent).
- Taking shortcuts to meet deadlines.
- Leaving dead code in place.

### Note

Over time, the technical debt must be paid back. Otherwise, the team's ability to fix issues and implement new features and enhancements will take longer and eventually become cost-prohibitive.

We have seen that technical debt adds a set of problems during development and makes it much more difficult to add extra customer value.

Having technical debt in a project saps productivity, frustrates development teams, makes code both hard to understand and fragile, increases the time to make changes and validate those changes. Unplanned work frequently gets in the way of planned work.

Longer-term, it also saps the organization's strength. Technical debt tends to creep up on an organization. It starts small and grows over time. Every time a quick hack is made or testing is circumvented because changes need to be rushed through, the problem grows worse and worse. Support costs get higher and higher, and invariably, a serious issue arises.

Eventually, the organization can't respond to its customers' needs in a timely and cost-efficient way.

## Automated measurement for monitoring

One key way to minimize the constant acquisition of technical debt is to use automated testing and assessment.

In the demos that follow, we'll look at one of the common tools used to assess the debt: SonarCloud. (The original on-premises version was SonarQube).

There are other tools available, and we'll discuss a few of them.

Later, in the next hands-on lab, you'll see how to configure your Azure Pipelines to use SonarCloud, understand the analysis results, and finally how to configure quality profiles to control the rule sets that are used by SonarCloud when analyzing your projects.

For more information, see [SonarCloud](#).

## To review:

Azure DevOps can be integrated with a wide range of existing tooling used to check code quality during builds.

Which code quality tools do you currently use (if any)?

What do you like or don't you like about the tools?

## Measure and manage technical debt

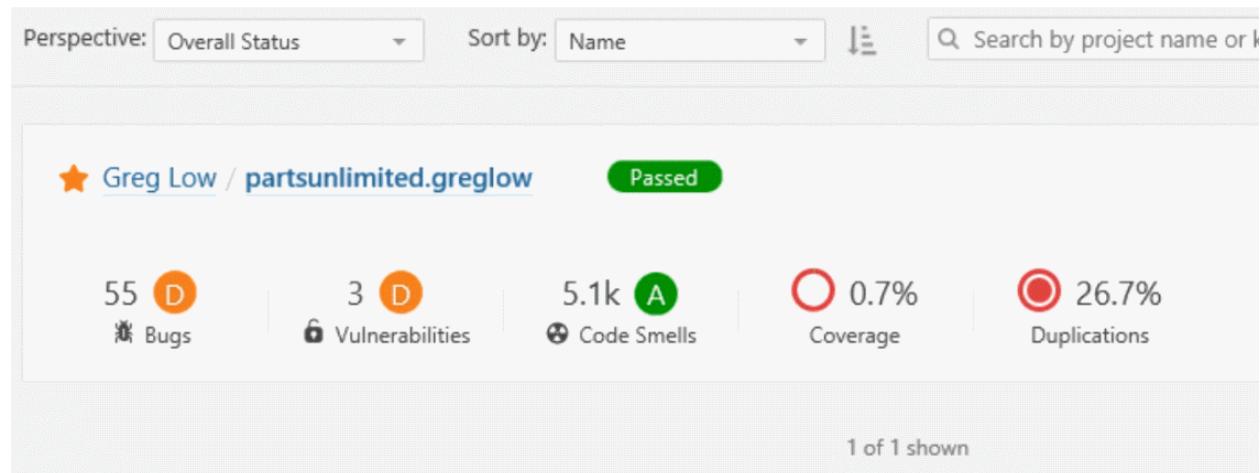
Completed 100 XP

- 1 minute

Integrating the assessment and measurement of technical debt and code quality overall is essential as part of your Continuous Integration and Deployment pipelines in Azure DevOps.

In the Continuous Integration course in this series, we showed how to add support for SonarCloud into an Azure DevOps pipeline.

After it's added and a build done, you can see an analysis of your code:



If you drill into the issues, you can see what the issues are, along with suggested remedies and estimates of the time required to apply a remedy.

Add a 'protected' constructor or the 'static' keyword to the class declaration. [...](#)

 Code Smell ▾  Major ▾  Open ▾ Not assigned ▾ 10min effort [Comment](#)

Refactor your code not to use hardcoded absolute paths or URIs. [...](#)

 Code Smell ▾  Minor ▾  Open ▾ Not assigned ▾ 20min effort [Comment](#)

Refactor your code not to use hardcoded absolute paths or URIs. [...](#)

 Code Smell ▾  Minor ▾  Open ▾ Not assigned ▾ 20min effort [Comment](#)

## Integrate other code quality tools

Completed 100 XP

- 1 minute

Many tools can be used to assess aspects of your code quality and technical debt.

### NDepend

For .NET developers, a common tool is NDepend.

NDepend is a Visual Studio extension that assesses the amount of technical debt a developer has added during a recent development period, typically in the last hour.

With this information, the developer might make the required corrections before ever committing the code.

NDepend lets you create code rules expressed as C# LINQ queries, but it has many built-in rules that detect a wide range of code smells.

### Resharper Code Quality Analysis

Resharper can make a code quality analysis from the command line. Also, be set to fail builds when code quality issues are found automatically.

Rules can be configured for enforcement across teams.

## Search in Azure DevOps

To find other code quality tools that are easy to integrate with Azure DevOps, search for the word **Quality** when adding a task into your build pipeline.

Add tasks | Refresh

Marketplace ^

- Resharper Code Quality Analysis**  
Run the Resharper Command-Line Tool and automatically fail builds when code quality issues are found. Configure team-shared coding rules for seamless code quality standards enforcement on each commit
- Build Quality Checks**  
Breaks a build based on quality metrics like number of warnings or code coverage.
- Code Quality NDepend for TFS 2017/TFS 2018 and VSTS**  
Build tasks and hub helping you to understand the technical debt of built code and it's evolution. Explore code metrics and define quality gates and trends
- Quality Gate Widget**  
Widget to show the SonarQube Quality Gate status for a project
- Quality Gate Web Smoke Test**  
Simple server-side (not agent based) Task for performing a smoke test on a released web application to validate that it is available at the end of a release as a Quality Gate.
- SonarCloud quality gate**  
this extension covers quality gate enforcement using data from SonarCloud
- SQL Enlight Code Quality**  
Build and Release tasks for SQL Enlight Code Quality integration with Visual Studio Team Services and Team Foundation Server 2017/2018.
- WhiteSource**  
Detect & fix security vulnerabilities, problematic open source licenses and quality issues.
- SonarQube build breaker**

For more information, you can see:

- [NDepend](#)

- Visual Studio marketplace
- ReSharper Code Quality Analysis

## Plan effective code reviews

Completed 100 XP

- 1 minute

Most developers would agree that code reviews can improve the quality of the applications they produce, but only if the process for the code reviews is effective. It is essential, upfront, to agree that everyone is trying to achieve better code quality.

Achieving code quality can seem to challenge because there's no single best way to write any piece of code, at least code with any complexity. Everyone wants to do good work and to be proud of what they create.

It means that it's easy for developers to become over-protective of their code. The organizational culture must let all involved feel that the code reviews are more like mentoring sessions where ideas about improving code are shared than interrogation sessions where the aim is to identify problems and blame the author.

The knowledge-sharing that can occur in mentoring-style sessions can be one of the most important outcomes of the code review process. It often happens best in small groups (even two people) rather than in large team meetings. And it's important to highlight what has been done well, not just what needs improvement.

Developers will often learn more in effective code review sessions than they'll in any formal training. Reviewing code should be an opportunity for all involved to learn, not just as a chore that must be completed as part of a formal process.

It's easy to see two or more people working on a problem and think that one person could have completed the task by themselves. That is a superficial view of the longer-term outcomes.

Team management needs to understand that improving the code quality reduces the cost of code, not increases it. Team leaders need to establish and foster an appropriate culture across their teams.

# Explore Git hooks

## Introduction to Git hooks

Completed 100 XP

- 4 minutes

Continuous delivery demands a significant level of automation. You can't be continuously delivering if you don't have a quality codebase. It's where git fares so well.

It lets you automate most of the checks in your codebase. Before committing the code into your local repository, let alone the remote.

## Git hooks

Git hooks are a mechanism that allows code to be run before or after certain Git lifecycle events.

For example, one could hook into the commit-msg event to validate that the commit message structure follows the recommended format.

The hooks can be any executable code, including shell, PowerShell, Python, or other scripts. Or they may be a binary executable. Anything goes!

The only criteria are that hooks must be stored in the .git/hooks folder in the repo root. Also, they must be named to match the related events (Git 2.x):

- applypatch-msg
- pre-applypatch
- post-applypatch
- pre-commit
- prepare-commit-msg
- commit-msg
- post-commit
- pre-rebase
- post-checkout
- post-merge
- pre-receive
- update

- post-receive
- post-update
- pre-auto-gc
- post-rewrite
- pre-push

## Practical use cases for using Git hooks

Since Git hooks execute the scripts on the specific event type they're called on, you can do much anything with Git hooks.

Some examples of where you can use hooks to enforce policies, ensure consistency, and control your environment:

- In Enforcing preconditions for merging
- Verifying work Item ID association in your commit message
- Preventing you & your team from committing faulty code
- Sending notifications to your team's chat room (Teams, Slack, HipChat, etc.)

In the next unit, you will see how to implement Git Hooks.

## Implement Git hooks

Completed 100 XP

- 5 minutes

When pushing quality into the development process, developing code locally, you want developers to identify and catch code quality issues. It's even before raising the pull request to trigger the branch policies.

Git hooks allow you to run custom scripts whenever certain important events occur in the Git life cycle—for example, committing, merging, and pushing.

Git ships with several sample hook scripts in the repo .git\hooks directory. Git executes the contents on the particular occasion type they're approached. You can do practically anything with Git hooks.

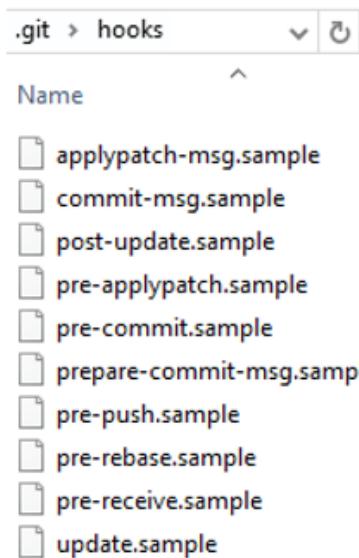
Here are a few instances of where you can use Git hooks to uphold arrangements, guarantee consistency, and control your environment:

- Enforcing preconditions for merging
- Verifying work Item ID association in your commit message Preventing you and your team from committing faulty code
- Sending notifications to your team's chat room (Teams, Slack, HipChat)

In this recipe, we'll use the pre-commit Git hook to scan the commit for keywords from a predefined list to block the commit if it contains any of these keywords.

## Getting ready

Let's start by exploring client-side Git hooks. Navigate to the repo .git\hooks directory – you'll find that there are a bunch of samples, but they're disabled by default.



## Automation through Source Control...

*"Using #Git hooks is like having little robot minions to carry out your every wish..."*

### Note

If you open that folder, you'll find a file called pre-commit.sample. To enable it, rename it to pre-commit by removing the .sample extension and making the script executable.

The script is found and executed when you attempt to commit using git commit. You commit successfully if your pre-commit script exits with a 0 (zero). Otherwise, the commit fails. If you're using Windows, simply renaming the file won't work.

Git will fail to find the shell in the chosen path specified in the script.

The problem is lurking in the first line of the script, the shebang declaration:

```
BashCopy  
#!/bin/sh
```

On Unix-like OSs, the `#!` Tells the program loader that it's a script to be interpreted, and `/bin/sh` is the path to the interpreter you want to use, `sh` in this case.

Windows isn't a Unix-like OS. Git for Windows supports Bash commands and shell scripts via Cygwin.

By default, what does it find when it looks for `sh.exe` at `/bin/sh`?

Nothing, nothing at all. Fix it by providing the path to the `sh` executable on your system. It's using the 64-bit version of Git for Windows, so the baseline looks like this:

```
BashCopy  
#!C:/Program\ Files/Git/usr/bin/sh.exe
```

## How to do it

How could Git hooks stop you from accidentally leaking Amazon AWS access keys to GitHub?

You can invoke a script at pre-commit.

Using Git hooks to scan the increment of code being committed into your local repository for specific keywords:

Replace the code in this pre-commit shell file with the following code.

```
BashCopy  
#!C:/Program\ Files/Git/usr/bin/sh.exe  
matches=$(git diff-index --patch HEAD | grep '^+' | grep -Pi  
'password|keyword2|keyword3')  
if [ ! -z "$matches" ]  
then  
    cat <<\EOT  
Error: Words from the blocked list were present in the diff:  
EOT  
    echo $matches  
    exit 1  
fi
```

You don't have to build the complete keyword scan list in this script.

You can branch off to a different file by referring to it here to encrypt or scramble if you want to.

## How it works

The Git diff-index identifies the code increment committed in the script. This increment is then compared against the list of specified keywords. If any matches are found, an error is raised to block the commit; the script returns an error message with the list of matches. The pre-commit script doesn't return 0 (zero), which means the commit fails.

## There's more

The repo .git\hooks folder isn't committed into source control. You may wonder how you share the goodness of the automated scripts you create with the team.

The good news is that, from Git version 2.9, you can now map Git hooks to a folder that can be committed into source control.

You could do that by updating the global settings configuration for your Git repository:

```
CmdCopy  
Git config --global core.hooksPath '~/.githooks'
```

If you ever need to overwrite the Git hooks you have set up on the client-side, you can do so by using the no-verify switch:

```
CmdCopy  
Git commit --no-verify
```

## Server-side service hooks with Azure Repos

So far, we've looked at the client-side Git Hooks on Windows. Azure Repos also exposes server-side hooks. Azure DevOps uses the exact mechanism itself to create Pull requests. You can read more about it at the [Server hooks event reference](#).

# Plan foster inner source

## Explore foster inner source

Completed 100 XP

- 2 minutes

The fork-based pull request workflow is popular with open-source projects because it allows anybody to contribute to a project.

You don't need to be an existing contributor or write access to a project to offer your changes.

This workflow isn't just for open source: forks also help support inner source workflows within your company.

Before forks, you could contribute to a project using Pull Requests.

The workflow is simple enough: push a new branch up to your repository, open a pull request to get a code review from your team, and have Azure Repos evaluate your branch policies.

You can click one button to merge your pull request into main and deploy when your code is approved.

This workflow is great for working on your projects with your team. But what if you notice a simple bug in a different project within your company and you want to fix it yourself?

What if you're going to add a feature to a project that you use, but another team develops?

It's where forks come in; forks are at the heart of inner source practices.

## Inner source

Inner source – sometimes called "internal open source" – brings all the benefits of open-source software development inside your firewall.

It opens your software development processes so that your developers can easily collaborate on projects across your company.

It uses the same processes that are popular throughout the open-source software communities.

But it keeps your code safe and secure within your organization.

**Microsoft uses the inner source approach heavily.**

As part of the efforts to standardize a one-engineering system throughout the company – backed by Azure Repos – Microsoft has also opened the source code to all our projects to everyone within the company.

Before the move to the inner source, Microsoft was "siloed": only engineers working on Windows could read the Windows source code.

Only developers working on Office could look at the Office source code.

So, if you're an engineer working on Visual Studio and you thought that you found a bug in Windows or Office – or wanted to add a new feature – you're out of luck.

But by moving to offer inner sources throughout the company, powered by Azure Repos, it's easy to fork a repository to contribute back.

As an individual making the change, you don't need to write access to the original repository, just the ability to read it and create a fork.

## Implement the fork workflow

Completed 100 XP

- 5 minutes

**A fork is a copy of a repository. Forking a repository allows you to experiment with changes without affecting the original project freely.**

Most commonly, forks are used to propose changes to someone else's project. Or use someone else's project as a starting point for your idea.

A fork is a complete copy of a repository, including all files, commits, and (optionally) branches.

Forks are a great way to support an Inner Source workflow: you can create a fork to suggest changes when you don't have permission to write to the original project directly.

Once you're ready to share those changes, it's easy to contribute them back-using pull requests.

## What's in a fork?

A fork starts with all the contents of its upstream (original) repository.

You can include all branches or limit them to only the default branch when you create a fork.

None of the permissions, policies, or build pipelines are applied.

The new fork acts as if someone cloned the original repository, then pushed it to a new, empty repository.

After a fork has been created, new files, folders, and branches aren't shared between the repositories unless a Pull Request (PR) carries them along.

## Sharing code between forks

You can create PRs in either direction: from fork to upstream or upstream to fork.

The most common approach will be from fork to upstream.

The destination repository's permissions, policies, builds, and work items will apply to the PR.

## Choosing between branches and forks

For a small team (2-5 developers), we recommend working in a single repo.

Everyone should work in a topic branch, and the main should be protected with branch policies.

As your team grows more significant, you may find yourself outgrowing this arrangement and prefer to switch to a forking workflow.

We recommend the forking workflow if your repository has many casual or infrequent committeees (like an open-source project).

Typically, only core contributors to your project have direct commit rights into your repository.

It would help if you asked collaborators from outside this core set of people to work from a fork of the repository.

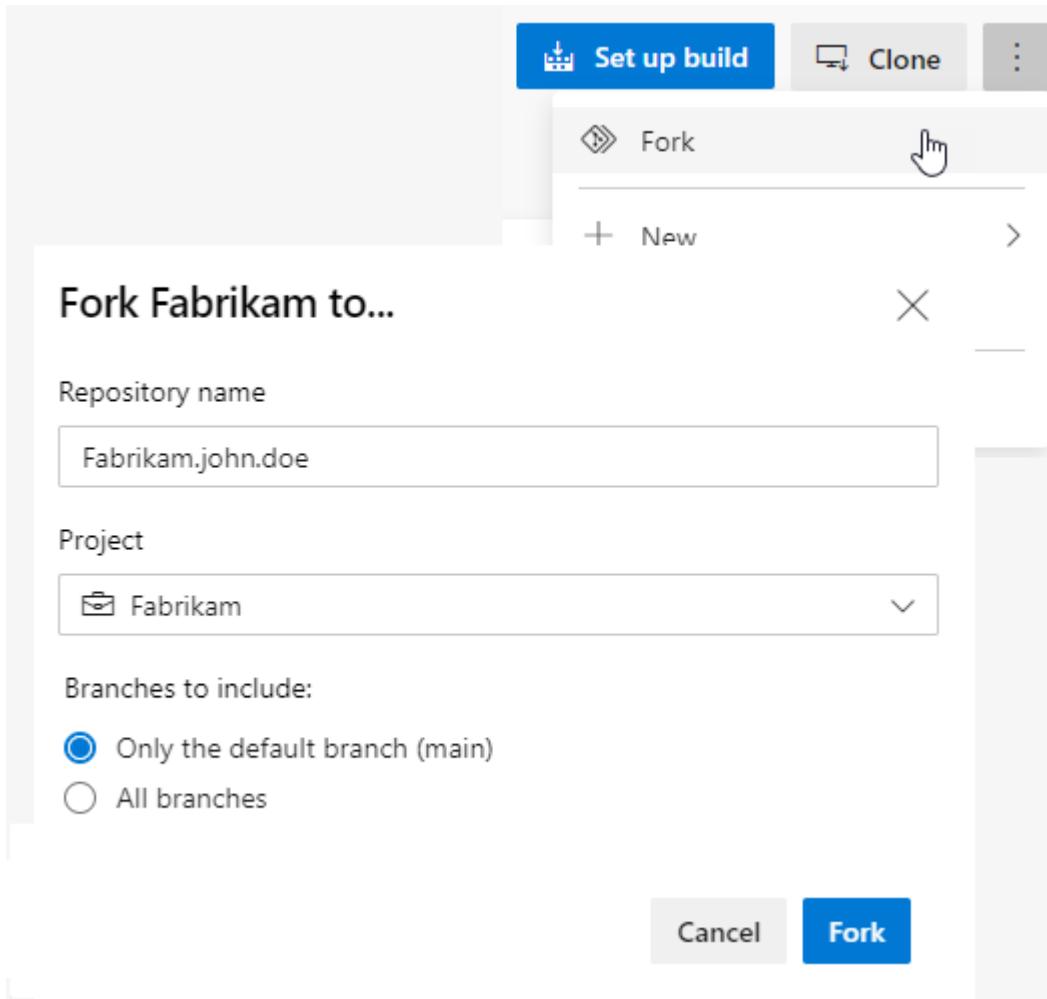
Also, it will isolate their changes from yours until you've had a chance to vet the work.

## The forking workflow

- Create a fork.
- Clone it locally.
- Make your changes locally and push them to a branch.
- Create and complete a PR to upstream.
- Sync your fork to the latest from upstream.

## Create the Fork

1. Navigate to the repository to fork and choose fork.
2. Specify a name and choose the project where you want the fork to be created. If the repository contains many topic branches, we recommend you fork only the default branch.
3. Choose the ellipsis, then Fork to create the fork.



## Note

You must have the Create Repository permission in your chosen project to create a fork. We recommend you create a dedicated project for forks where all contributors have the Create Repository permission. For an example of granting this permission, see Set Git repository permissions.

## Clone your fork locally

Once your fork is ready, clone it using the command line or an IDE like Visual Studio. The fork will be your origin remote.

For convenience, after cloning, you'll want to add the upstream repository (where you forked from) as a remote named upstream.

CmdCopy

```
git remote add upstream {upstream_url}
```

## Make and push changes

It's possible to work directly in main - after all, this fork is your copy of the repo.

We recommend you still work in a topic branch, though.

It allows you to maintain multiple independent workstreams simultaneously.

Also, it reduces confusion later when you want to sync changes into your fork.

Make and commit your changes as you normally would. When you're done with the changes, push them to origin (your fork).

## Create and complete a PR

Open a pull request from your fork to the upstream. All the policies required reviewers and builds will be applied in the upstream repo. Once all policies are satisfied, the PR can be completed, and the changes become a permanent part of the upstream repo.

The screenshot shows the GitHub fork interface for the repository 'FabrikamFiber / Fabrika...'. The user is creating a new pull request from their fork 'FabrikamFiber.jamal.fork' into the 'master' branch of the upstream repository 'FabrikamFiber'. The pull request title is 'Added a new option to the settings page'. The description includes a note about adding delivery preferences and three checkboxes for opting out of email, opting into email, and specifying an alternate address. The interface also shows options for reviewers and work items.

New Pull Request

FabrikamFiber.jamal.fork into FabrikamFiber master

Title \*

Added a new option to the settings page

Add label

Description

Added a new option to the settings page.

Added a new option for users to manage delivery preferences. Tested the following:

- [ ] Opt out of email
- [ ] Opt into email
- [ ] Specify an alternate address

Markdown supported.

Aa **I** ↲ ⌂ ⌂ ⌂ @ # ⌂

Added a new option to the settings page.

Added a new option for users to manage delivery preferences. Tested the following:

- Opt out of email
- Opt into email
- Specify an alternate address

Reviewers

[FabrikamFiber]\FabrikamFiber Team X Search users and groups to add as reviewers

Work Items

## Important

Anyone with the Read permission can open a PR to upstream. If a PR build pipeline is configured, the build will run against the code introduced in the fork.

## Sync your fork to the latest

When you've gotten your PR accepted into upstream, you'll want to make sure your fork reflects the latest state of the repo.

We recommend rebasing on upstream's main branch (assuming main is the main development branch).

#### CmdCopy

```
git fetch upstream main  
git rebase upstream/main  
git push origin
```

The forking workflow lets you isolate changes from the main repository until you're ready to integrate them. When you're ready, integrating code is as easy as completing a pull request.

## Describe inner source with forks

Completed 100 XP

- 4 minutes

People fork repositories when they want to change the code in a repository they don't have to write access to.

If you don't have write access, you aren't part of the team contributing to that repository, so why would you modify the code repository?

We tend to look for technical reasons to improve something in our work.

You may find a better way to implement the solution or enhance functionality by contributing to or improving an existing feature.

You can fork repositories in the following situations:

- I want to make a change.
- I think the project is exciting and may wish to use it.
- I want to use some code in that repository as a starting point for my project.

Software teams are encouraged to contribute to all projects internally, not just their software projects.

Forks are a great way to foster a culture of inner open source.

Forks are a recent addition to the Azure DevOps Git repositories.

This recipe will teach you to fork an existing repository and contribute changes upstream via a pull request.

## Getting ready

A fork starts with all the contents of its upstream (original) repository.

When you create a fork in Azure DevOps, you can include all branches or limit them to only the default branch.

A fork doesn't copy the permissions, policies, or build definitions of the repository being forked.

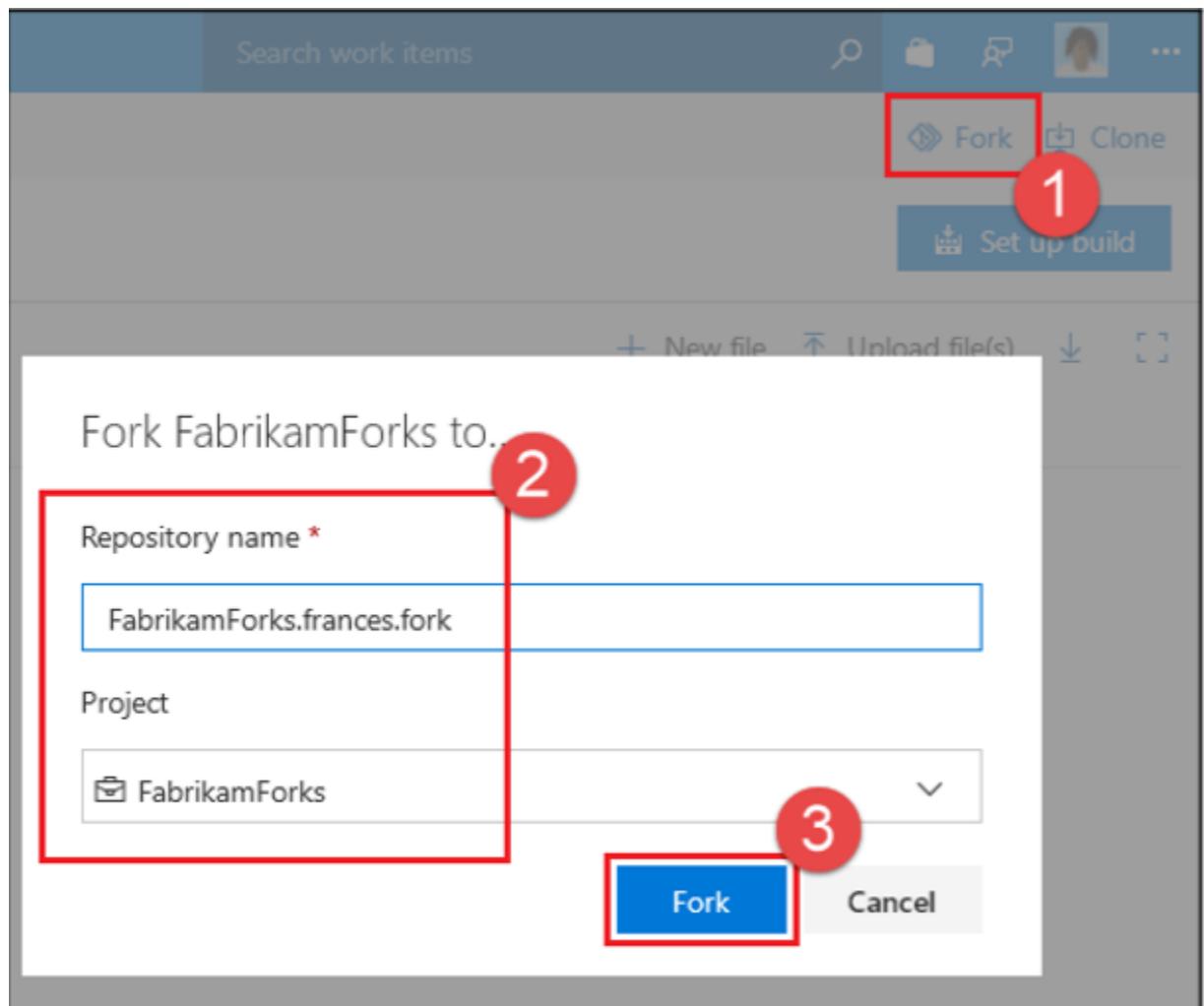
After a fork has been created, the newly created files, folders, and branches aren't shared between the repositories unless you start a pull request.

Pull requests are supported in either direction: from fork to upstream or upstream to fork.

The most common approach for a pull request will be from fork to upstream.

## How to do it

1. Choose the Fork button (1), then select the project where you want the fork to be created (2). Give your fork a name and choose the Fork button (3).



2. Once your fork is ready, clone it using the command line or an IDE, such as Visual Studio. The fork will be your origin remote. For convenience, you'll want to add the upstream repository (where you forked from) as a remote named upstream. On the command line, type:

```
CMDCopy  
git remote add upstream {upstream_url}
```
3. It's possible to work directly in the main – this fork is your copy of the repo. We recommend you still work in a topic branch, though. It allows you to maintain multiple independent workstreams simultaneously. Also, it reduces confusion later when you want to sync changes into your fork. Make and commit your changes as you normally would. When you finish the modifications, push them to the origin (your fork).
4. Open a pull request from your fork to the upstream. The upstream repo will apply all the policies required for reviewers and builds. Once all the policies

are satisfied, the PR can be completed, and the changes become a permanent part of the upstream repo:

The screenshot shows a pull request interface. At the top, there's a 'Description' field containing a commit message: '# Updated HomeController.cs + \_\_Added\_\_ a new method for concatenating two strings'. Below this is a 'Markdown supported.' note and a rich text editor toolbar with icons for bold, italic, etc. The main title of the PR is 'Updated HomeController.cs'. A bullet point under the title says '• Added a new method for concatenating two strings'. The 'Reviewers' section shows '[PartsUnlimited]\\PartsUnlimited Team' selected. The 'Work Items' section shows a search bar with 'Search work items by ID or title' and a result '38 Retrieve the user profile for personalization settings'.

- When your PR is accepted upstream, you must ensure your fork reflects the latest repo state. We recommend rebasing on the upstream's main branch (assuming the main is the main development branch). On the command line, run:

```
CMDCopy  
git fetch upstream main  
git rebase upstream/main  
git push origin
```

For more information about Git, see:

- [Clone an Existing Git repo.](#)
- [Azure Repos Git Tutorial.](#)

# Manage Git repositories

## Work with large repositories

Completed 100 XP

- 3 minutes

Git is a great version control system widely adopted and recommended, but a few concerns should be made and taken care of when working with large repositories.

While having a local copy of repositories in a distributed version control system is functional, that can be a significant problem when large repositories are in place.

For example, Microsoft discovered this issue when migrating a repository with over 300 GB of data from an internal system to Git.

### Why repositories become large

There are two primary causes for large repositories:

- Long history
- Large binary files

### Shallow clone

If developers don't need all the available history in their local repositories, a good option is to implement a shallow clone.

It saves both space on local development systems and the time it takes to sync.

You can specify the depth of the clone that you want to execute:

DOSCopy

```
git clone --depth [depth] [clone-url]
```

You can also reduce clones by filtering branches or cloning only a single branch.

## VFS for Git

VFS for Git helps with large repositories. It requires a Git LFS client.

Typical Git commands are unaffected, but the Git LFS works with the standard filesystem to download necessary files in the background when you need files from the server.

The Git LFS client was released as open-source. The protocol is a straightforward one with four endpoints similar to REST endpoints.

For more information on large repositories, see: [Working with large files](#) and [Virtual File System for Git: Enable Git at Enterprise Scale](#).

## Scalar



Scalar is a .NET Core application available for Windows and macOS. With tools and extensions for Git to allow very large repositories to maximize your Git command performance. Microsoft uses it for Windows and Office repositories.

If Azure Repos hosts your repository, you can clone a repository using the [GVFS protocol](#).

It achieves by enabling some advanced Git features, such as:

- **Partial clone:** reduces time to get a working repository by not downloading all Git objects right away.
- **Background prefetch:** downloads Git object data from all remotes every hour, reducing the time for foreground git fetch calls.

- *Sparse-checkout*: limits the size of your working directory.
- *File system monitor*: tracks the recently modified files and eliminates the need for Git to scan the entire work tree.
- *Commit-graph*: accelerates commit walks and reachability calculations, speeding up commands like `git log`.
- *Multi-pack-index*: enables fast object lookups across many pack files.
- *Incremental repack*: Repacks the packed Git data into fewer pack files without disrupting concurrent commands using the multi-pack-index.

## Note

We update the list of features that Scalar automatically configures as a new Git version is released.

For more information, see:

- [microsoft/scalar: Scalar](#).
- [Introducing Scalar: Git at scale for everyone](#).

# Purge repository data

Completed 100 XP

- 2 minutes

While one of the benefits of Git is its ability to hold long histories for repositories efficiently, there are times when you need to purge data.

The most common situations are where you want to:

- Significantly reduce the size of a repository by removing history.
- Remove a large file that was accidentally uploaded.
- Remove a sensitive file that shouldn't have been uploaded.

If you commit sensitive data (for example, password, key) to Git, it can be removed from history. Two tools are commonly used:

## git filter-repo tool

The git filter-repo is a tool for rewriting history.

Its core filter-repo contains a library for creating history rewriting tools. Users with specialized needs can quickly create entirely new history rewriting tools.

## Note

More details are in the repository [git-filter-repo](#).

## BFG Repo-Cleaner

BFG Repo-Cleaner is a commonly used open-source tool for deleting or "fixing" content in repositories. It's easier to use than the git filter-branch command. For a single file or set of files, use the **--delete-files** option:

BashCopy

```
$ bfg --delete-files file_I_should_not_have_committed
```

The following bash shows how to find all the places that a file called passwords.txt exists in the repository. Also, to replace all the text in it, you can execute the **--replace-text** option:

BashCopy

```
$ bfg --replace-text passwords.txt
```

For more information, see:

[Quickly rewrite git repository history.](#)

[Removing files from Git Large File Storage.](#)

[Removing sensitive data from a repository.](#)

[BFG Repo Cleaner.](#)

## Manage releases with GitHub Repos

Completed 100 XP

- 3 minutes

Part of the release process starts with your version control. You'll understand how to manage releases in the repository using GitHub.

In the following modules, you'll see details about deploying a piece of software after packaging your code, binary files, release notes, and related tasks.

Releases in GitHub are based on [Git tags](#). You can think of a tag as a photo of your repository's current state. If you need to mark an essential phase of your code or your following deliverable code is done, you can create a tag and use it during the build and release process to package and deploy that specific version. For more information, see [Viewing your repository's releases and tags](#).

When creating new releases with release notes, it's possible to @mentions contributors, add links to binary files and edit or delete existing releases.

The screenshot shows a GitHub release page for version v2.198.1, which is a pre-release. The page includes the following sections:

- Features:**
  - [macOS] Return the correct minor version on 11 and later OS. (#3605)
  - Add Event Log dumping (#3653)
  - Add passive validation - list local group memberships (#3673)
  - Added RepoType telemetry for checkout task (#3677)
  - Enabling validation of checksum for online agent update for ADO OnPrem (#3679)
  - Add function to read waagent.conf settings and condition to run it only on Linux (#3680)
  - Add passive validation - dumping cloud-init logs (#3681)
  - Added masking for environment variables containing credentials in diagnostic logs (#3682)
  - Download TEE plugin conditionally during checkout (#3684)
  - Added information about user groups into environment file (Linux, MacOS) (#3690)
  - Add cloud-init logs to diagnostics archive (#3700)
- Bugs:**
  - Porting logic of handling negative patterns for DownloadBuildArtifacts task (#3664)
  - Added tests for ported logic of handling negative patterns for DownloadBuildArtifacts task (#3665)
  - Adding support of negative patterns for DownloadBuildArtifacts task in scenarios with using file share (#3666)
  - Fix permissions setting on MacOS while downloading TEE (#3704)
- Agent Downloads:**

|             | Package                             | SHA-256  |
|-------------|-------------------------------------|--|
| Windows x64 | vsts-agent-win-x64-2.198.1.zip      | 8c426cc43d23d709e4540b6152c57a08394738ee302e8a92c5f4f763da93feef |
| Linux x64   | vsts-agent-linux-x64-2.198.1.tar.gz | 0da3ac2dc74a271dc6718c0aa6057effa58280a8191bf20ad165d57810b42d9f |
- Assets:** 3
  - assets.json
  - Source code (zip)
  - Source code (tar.gz)

*Image reference: [Releases · Microsoft/azure-pipelines-agent \(github.com\)](#)*

Also, you can:

- Publish an action from a specific release in GitHub Marketplace.
- Choose whether Git LFS objects are included in the ZIP files and tarballs GitHub creates for each release.
- Receive notifications when new releases are published in a repository.

## Creating a release

To create a release, use the `gh release create` command. Replace the **tag** with the desired tag name for the release and follow the interactive prompts.

Copy

```
gh release create tagname
```

To create a prerelease with the specified title and notes.

Copy

```
gh release create v1.2.1 --title
```

If you @mention any GitHub users in the notes, the published release on GitHub.com will include a Contributors section with an avatar list of all the mentioned users.

You can check other commands and arguments from the [GitHub CLI manual](#).

## Editing a release

You can't edit Releases with GitHub CLI.

To edit, use the Web Browser:

1. Navigate to the main repository page on GitHub.com.
2. Click **Releases** to the right of the list of files.
3. Click on the **edit icon** on the right side of the page, next to the release you want to edit.
4. Edit the details for the release, then click **Update release**.

## Deleting a release

To delete a release, use the following command, replace the **tag** with the release tag to delete, and use the -y flag to skip confirmation.

Copy

```
gh release delete tag -y
```

For more information, see:

- [Managing releases in a repository - GitHub Docs](#) - If you want to perform the same steps from Web Browser instead of GitHub CLI.
- [Publishing an action in the GitHub Marketplace](#).
- [Managing Git LFS objects in archives of your repository](#).
- [Viewing your subscriptions](#).

## Automate release notes with GitHub

Completed 100 XP

- 2 minutes

After learning how to create and manage release tags in your repository, you'll learn how to configure the automatically generated release notes template from your GitHub releases.

You can generate an overview of the contents of a release, and you can also customize your automated release notes.

It's possible to use labels to create custom categories to organize pull requests you want to include or exclude specific labels and users from appearing in the output.

## Creating automatically generated release notes

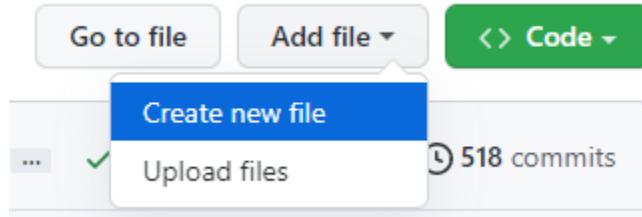
While configuring your release, you'll see the option Auto-generate release notes to include all changes between your tag and the last release. If you never created a release, it will consist of all changes from your repository.

You can choose if you want to customize it or leave it as it is.

# Configuring automatically generated release notes template

You can customize the auto-generate release notes template by using the following steps.

1. Navigate to your repository and create a new file.



2. You can use the name **.github/release.yml** to create the **release.yml** file in the **.github** directory.

A screenshot of the GitHub release configuration editor. At the top, there's a breadcrumb navigation: 'devops-journey / .github / release.yml' followed by a 'in main' indicator. Below this is a toolbar with 'Edit new file' and 'Preview' buttons. The 'Preview' button has an eye icon. The main area shows a single line of YAML code: '1'.

```
1
```

3. Specify in YAML the pull request labels and authors you want to exclude from this release. You can also create new categories and list the pull request labels in each. For more information about configuration options, see [Automatically generated release notes - GitHub Docs](#).

#### Example configuration:

YMLCopy

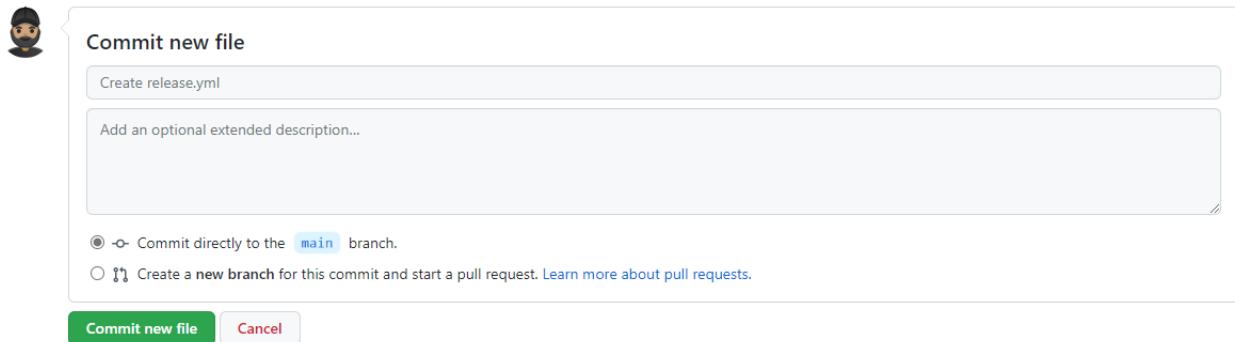
```
# .github/release.yml

changelog:
  exclude:
    labels:
      - ignore-for-release
    authors:
      - octocat
  categories:
    - title: Breaking Changes ❌
      labels:
        - Semver-Major
        - breaking-change
    - title: Exciting New Features 🎉
      labels:
        - Semver-Minor
        - enhancement
    - title: Other Changes
```

**labels:**

- \*

4. Commit your new file.



5. Try to create a new release and click + **Auto-generate release notes** to see the template structure.

For more information, see:

- [About releases - GitHub Docs](#)
- [Linking to releases - GitHub Docs](#)
- [Automation for release forms with query parameters - GitHub Docs](#)

# Explore Azure Pipelines

## Explore the concept of pipelines in DevOps

Completed 100 XP

- 4 minutes

Business demands continuous delivery of value. Value is created only when a product is delivered to a satisfied customer.

It's not created when one silo in the process is completed.

It demands that you reset focus from silos to an end-to-end flow of value.

The core idea is to create a repeatable, reliable, and incrementally-improving process for taking software from concept to customer.

The goal is to enable a constant flow of changes into production via an automated software production line.

Think of it as a pipeline. The pipeline breaks down the software delivery process into stages.

Each stage aims to verify the quality of new features from a different angle to validate the new functionality and prevent errors from affecting your users.

The pipeline should provide feedback to the team. Also, visibility into the changes flows to everyone involved in delivering the new feature(s).

A delivery pipeline enables the flow of more minor changes more frequently, with a focus on flow.

Your teams can concentrate on optimizing the delivery of changes that bring quantifiable value to the business.

This approach leads teams to continuously monitor and learn where they're finding obstacles, resolve those issues, and gradually improve the pipeline's flow.

As the process continues, the feedback loop provides new insights into new issues and barriers to be resolved.

The pipeline is the focus of your continuous improvement loop.

A typical pipeline will include the following stages: build automation and continuous integration, test automation, and deployment automation.

## **Build automation and continuous integration**

The pipeline starts by building the binaries to create the deliverables passed to the following stages. New features implemented by the developers are integrated into the central code base, built, and unit tested. It's the most direct feedback cycle that informs the development team about the health of their application code.

## **Test automation**

The new version of an application is rigorously tested throughout this stage to ensure that it meets all wished system qualities. It's crucial that all relevant aspects—whether functionality, security, performance, or compliance—are verified by the pipeline. The stage may involve different types of automated or (initially, at least) manual activities.

## **Deployment automation**

A deployment is required every time the application is installed in an environment for testing, but the most critical moment for deployment automation is rollout time.

Since the preceding stages have verified the overall quality of the system, it's a low-risk step.

The deployment can be staged, with the new version being initially released to a subset of the production environment and monitored before being rolled out.

The deployment is automated, allowing for the reliable delivery of new functionality to users within minutes if needed.

## Your pipeline needs platform provisioning and configuration management

The deployment pipeline is supported by platform provisioning and system configuration management. It allows teams to create, maintain, and tear down complete environments automatically or at the push of a button.

Automated platform provisioning ensures that your candidate applications are deployed to, and tests carried out against correctly configured and reproducible environments.

It also helps horizontal scalability and allows the business to try out new products in a sandbox environment at any time.

## Orchestrating it all: release and pipeline orchestration

The multiple stages in a deployment pipeline involve different groups of people collaborating and supervising the release of the new version of your application.

Release and pipeline orchestration provide a top-level view of the entire pipeline, allowing you to define and control the stages and gain insight into the overall software delivery process.

By carrying out value stream mappings on your releases, you can highlight any remaining inefficiencies and hot spots and pinpoint opportunities to improve your pipeline.

These automated pipelines need infrastructure to run on. The efficiency of this infrastructure will have a direct impact on the effectiveness of the pipeline.

## Describe Azure Pipelines

Completed 100 XP

- 3 minutes

Azure Pipelines is a cloud service that automatically builds and tests your code project and makes it available to other users. It works with just about any language or project type.

Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to test and build your code and ship it to any target constantly and consistently.

## Does Azure Pipelines work with my language and tools?

Azure Pipelines is a fully featured cross-platform CI and CD service. It works with your preferred Git provider and can deploy to most major cloud services, including Azure services.

## Languages

You can use many languages with Azure Pipelines, such as Python, Java, PHP, Ruby, C#, and Go.

## Version control systems

Before you use continuous integration and continuous delivery practices for your applications, you must have your source code in a version control system. Azure Pipelines integrates with GitHub, GitLab, Azure Repos, Bitbucket, and Subversion.

## Application types

You can use Azure Pipelines with most application types, such as Java, JavaScript, Python, .NET, PHP, Go, XCode, and C++.

## Deployment targets

Use Azure Pipelines to deploy your code to multiple targets. Targets including:

- Container registries.
- Virtual machines.
- Azure services, or any on-premises or cloud target such:
  - Microsoft Azure.
  - Google Cloud.

- Amazon Web Services (AWS).

## Package formats

To produce packages that others can consume, you can publish NuGet, npm, or Maven packages to the built-in package management repository in Azure Pipelines.

You also can use any other package management repository of your choice.

## Why should I use CI and CD, and Azure Pipelines?

Implementing CI and CD pipelines help to ensure consistent and quality code that's readily available to users.

Azure Pipelines is a quick, easy, and safe way to automate building your projects and making them available to users.

## Use CI and CD for your project

Continuous integration is used to automate tests and builds for your project. CI helps to catch bugs or issues early in the development cycle when they're easier and faster to fix. Items known as artifacts are produced from CI systems. The continuous delivery release pipelines use them to drive automatic deployments.

Continuous delivery is used to automatically deploy and test code in multiple stages to help drive quality. Continuous integration systems produce deployable artifacts, which include infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fixes to the target of your choice.

### Continuous integration (CI)

### Continuous delivery (CD)

Increase code coverage.

Automatically deploy code to production.

Build faster by splitting test and build runs.

Ensure deployment targets have the latest code.

Automatically ensure you don't ship broken code.

Use tested code from the CI process.

Run tests continually.

## Use Azure Pipelines for CI and CD

There are several reasons to use Azure Pipelines for your CI and CD solution. You can use it to:

- Work with any language or platform.
- Deploy to different types of targets at the same time.
- Integrate with Azure deployments.
- Build on Windows, Linux, or macOS machines.
- Integrate with GitHub.
- Work with open-source projects.

## Understand Azure Pipelines key terms

Completed 100 XP

- 4 minutes

Understanding the basic terms and parts of Azure Pipelines helps you further explore how it can help you deliver better code more efficiently and reliably.

### Agent

When your build or deployment runs, the system begins one or more jobs. An agent is installable software that runs a build or deployment job.

### Artifact

An artifact is a collection of files or packages published by a build. Artifacts are made available for the tasks, such as distribution or deployment.

## Build

A build represents one execution of a pipeline. It collects the logs associated with running the steps and the test results.

## Continuous delivery

Continuous delivery (CD) (also known as Continuous Deployment) is a process by which code is built, tested, and deployed to one or more test and production stages.

Deploying and testing in multiple stages helps drive quality. Continuous integration systems produce deployable artifacts, which include infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fix existing systems. Monitoring and alerting systems constantly run to drive visibility into the entire CD process. This process ensures that errors are caught often and early.

## Continuous integration

Continuous integration (CI) is the practice used by development teams to simplify the testing and building of code. CI helps to catch bugs or problems early in the development cycle, making them more accessible and faster to fix. Automated tests and builds are run as part of the CI process. The process can run on a schedule, whenever code is pushed, or both. Items known as artifacts are produced from CI systems. The continuous delivery release pipelines use them to drive automatic deployments.

## Deployment target

A deployment target is a virtual machine, container, web app, or any service used to host the developed application. A pipeline might deploy the app to one or more deployment targets after the build is completed and tests are run.

## Job

A build contains one or more jobs. Most jobs run on an agent. A job represents an execution boundary of a set of steps. All the steps run together on the same agent.

For example, you might build two configurations - x86 and x64. In this case, you have one build and two jobs.

## Pipeline

A pipeline defines the continuous integration and deployment process for your app. It's made up of steps called tasks.

It can be thought of as a script that describes how your test, build, and deployment steps are run.

## Release

When you use the visual designer, you can create a release or a build pipeline. A release is a term used to describe one execution of a release pipeline. It's made up of deployments to multiple stages.

## Stage

Stages are the primary divisions in a pipeline: "build the app," "run integration tests," and "deploy to user acceptance testing" are good examples of stages.

## Task

A task is the building block of a pipeline. For example, a build pipeline might consist of build and test tasks. A release pipeline consists of deployment tasks. Each task runs a specific job in the pipeline.

## Trigger

A trigger is set up to tell the pipeline when to run. You can configure a pipeline to run upon a push to a repository at scheduled times or upon completing another build. All these actions are known as triggers.

# Design a container build strategy

## Examine structure of containers

Completed 100 XP

- 5 minutes

If you're a programmer or techie, you've at least heard of Docker: a helpful tool for packing, shipping, and running applications within "containers."

With all the attention it's getting these days, it would be hard not to, from developers and system admins alike.

There's a difference between containers and Docker. A container is a thing that runs a little program package, while Docker is the container runtime and orchestrator.

## What are containers, and why do you need them?

Containers are a solution to the problem of how to get the software to run reliably when moved from one computing environment to another.

It could be from a developer's laptop to a test environment, from a staging environment to production. Also, from a physical machine in a data center to a VM in a private or public cloud.

Problems arise when the supporting software environment isn't identical.

For example, say you'll develop using Python 3, but when it gets deployed to production, it will run on Python 2.7. It's likely to cause several issues.

It's not limited to the software environment; you're likely to come across issues in production if there are differences in the networking stack between the two environments.

## How do containers solve this problem?

A container consists of an entire runtime environment:

- An application, plus all its dependencies.
- Libraries and other binaries.
- Configuration files needed to run it, bundled into one package.

You can resolve it by containerizing the application platform and its dependencies. Also, differences in OS distributions and underlying infrastructure are abstracted.

## What's the difference between containers and virtualization?

Containers and VMs are similar in their goals: to isolate an application and its dependencies into a self-contained unit that can run anywhere. They remove the need for physical hardware, allowing for:

- More efficient use of computing resources.
- Energy consumption.
- Cost-effectiveness.

The main difference between containers and VMs is in their architectural approach. Let's take a closer look.

### Virtual Machines

A VM is essentially an emulation of a real computer that executes programs like a real computer. VMs run on top of a physical machine using a "hypervisor."

As you can see in the diagram, VMs package up the virtual hardware, a kernel (OS), and user space for each new VM.

### Container

Unlike a VM, which provides hardware virtualization, a container provides operating-system-level virtualization by abstracting the "user space."

This diagram shows that containers package up just the user space, not the kernel or virtual hardware like a VM does. Each container gets its isolated user space to allow multiple containers to run on a single host machine. We can see that all the operating

system-level architecture is being shared across containers. The only parts that are created from scratch are the bins and libs. It's what makes containers so lightweight.

## Work with Docker containers

Completed 100 XP

- 2 minutes

### Container Lifecycle

The standard steps when working with containers are:

**Docker build** - You create an image by executing a Dockerfile.

**Docker pull** - You retrieve the image, likely from a container registry.

**Docker run** - You execute the container. An instance is created of the image.

You can often execute the docker run without needing first to do the docker pull.

In that case, Docker will pull the image and then run it. Next time, it won't need to pull it again.

## Understand Dockerfile core concepts

Completed 100 XP

- 4 minutes

Dockerfiles are text files that contain the commands needed by **docker build** to assemble an image.

Here's an example of a basic Dockerfile:

Copy

```
FROM ubuntu
LABEL maintainer="johndoe@contoso.com"
ADD appsetup /
```

```
RUN /bin/bash -c 'source $HOME/.bashrc; \  
echo $HOME'  
CMD ["echo", "Hello World from within the container"]
```

The first line refers to the parent image based on which this new image will be based.

Generally, all images will be based on another existing image. In this case, the Ubuntu image would be retrieved from either a local cache or from DockerHub.

An image that doesn't have a parent is called a **base** image. In that rare case, the FROM line can be omitted, or **FROM scratch** can be used instead.

The second line indicates the email address of the person who maintains this file. Previously, there was a MAINTAINER command, but that has been deprecated and replaced by a label.

The third line adds a file to the root folder of the image. It can also add an executable.

The fourth and fifth lines are part of a RUN command. Note the use of the backslash to continue the fourth line onto the fifth line for readability. It's equivalent to having written it instead:

Copy

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

The RUN command is run when the docker build creates the image. It's used to configure items within the image.

By comparison, the last line represents a command that will be executed when a new container is created from the image; it's run after container creation.

For more information, you can see:

[Dockerfile reference](#)

## Examine multi-stage dockerfiles

Completed 100 XP

- 4 minutes

What are multi-stage Dockerfiles? Multi-stage builds give the benefits of the builder pattern without the hassle of maintaining three separate files.

Let us look at a multi-stage Dockerfile.

```
dockerCopy
FROM mcr.microsoft.com/dotnet/core/aspnetcore:3.1 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build
WORKDIR /src
COPY ["WebApplication1.csproj", ""]
RUN dotnet restore "./WebApplication1.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "WebApplication1.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "WebApplication1.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

At first, it simply looks like several dockerfiles stitched together. Multi-stage Dockerfiles can be layered or inherited.

When you look closer, there are a couple of key things to realize.

Notice the third stage.

```
FROM build AS publish
```

build isn't an image pulled from a registry. It's the image we defined in stage 2, where we named the result of our-build (SDK) image "builder." Docker build will create a named image we can later reference.

We can also copy the output from one image to another. It's the real power to compile our code with one base SDK image (`mcr.microsoft.com/dotnet/core/sdk:3.1`) while creating a production image based on an optimized runtime image (`mcr.microsoft.com/dotnet/core/aspnet:3.1`). Notice the line.

```
COPY --from=publish /app/publish .
```

It takes the /app/publish directory from the published image and copies it to the working directory of the production image.

## Breakdown of stages

The first stage provides the base of our optimized runtime image. Notice it derives from `mcr.microsoft.com/dotnet/core/aspnet:3.1`.

We would specify extra production configurations, such as registry configurations, MSexec of other components. You would hand off any of those environment configurations to your ops folks to prepare the VM.

The second stage is our build environment. `mcr.microsoft.com/dotnet/core/sdk:3.1` This includes everything we need to compile our code. From here, we have compiled binaries we can publish or test—more on testing in a moment.

The third stage derives from our build stage. It takes the compiled output and "publishes" them in .NET terms.

Publish means taking all the output required to deploy your "app/publish/service/component" and placing it in a single directory. It would include your compiled binaries, graphics (images), JavaScript, and so on.

The fourth stage takes the published output and places it in the optimized image we defined in the first stage.

## Why is publish separate from the build?

You'll likely want to run unit tests to verify your compiled code. Or the aggregate of the compiled code from multiple developers being merged continues to function as expected.

You could place the following stage between builder and publish to run unit tests.

```
dockerCopy
FROM build AS test
WORKDIR /src/Web.test
RUN dotnet test
```

If your tests fail, the build will stop to continue.

## Why is base first?

You could argue it's simply the logical flow. We first define the base runtime image. Get the compiled output ready, and place it in the base image.

However, it's more practical. While debugging your applications under Visual Studio Container Tools, VS will debug your code directly in the base image.

When you hit F5, Visual Studio will compile the code on your dev machine. The first stage then volume mounts the output to the built runtime image.

You can test any configurations you have made to your production image, such as registry configurations or otherwise.

When the `docker build --target base` is executed, docker starts processing the dockerfile from the beginning through the stage (target) defined.

Since the base is the first stage, we take the shortest path, making the F5 experience as fast as possible.

If the base were after compilation (builder), you would have to wait for all the next steps to complete.

One of the perf optimizations we make with VS Container Tools is to take advantage of the Visual Studio compilations on your dev machine.

## Examine considerations for multiple stage builds

Completed 100 XP

- 3 minutes

## Adopt container modularity

Try to avoid creating overly complex container images that couple together several applications.

Instead, use multiple containers and try to keep each container to a single purpose.

The website and the database for a web application should likely be in separate containers.

There are always exceptions to any rule but breaking up application components into separate containers increases the chances of reusing containers.

It also makes it more likely that you could scale the application.

For example, in the web application mentioned, you might want to add replicas of the website container but not for the database container.

## Avoid unnecessary packages

To help minimize image sizes, it's also essential to avoid including packages that you suspect might be needed but aren't yet sure if they're required.

Only include them when they're required.

## Choose an appropriate base

While optimizing the contents of your Dockerfiles is essential, it's also crucial to choose the appropriate parent (base) image. Start with an image that only contains packages that are required.

## Avoid including application data

While application data can be stored in the container, it will make your images more prominent.

It would be best to consider using **docker volume** support to maintain the isolation of your application and its data. Volumes are persistent storage mechanisms that exist outside the lifespan of a container.

For more information, see [Use multiple-stage builds](#).

# Explore Azure container-related services

Completed 100 XP

- 4 minutes

Azure provides a wide range of services that help you work with containers.

Here are the essential services that are involved:

## [Azure Container Instances \(ACI\)](#)

Running your workloads in Azure Container Instances (ACI) allows you to create your applications rather than provisioning and managing the infrastructure that will run the applications.

ACIs are simple and fast to deploy, and when you're using them, you gain the security of hypervisor isolation for each container group. It ensures that your containers aren't sharing an operating system kernel with other containers.

## [Azure Kubernetes Service \(AKS\)](#)

Kubernetes has quickly become the de-facto standard for container orchestration. This service lets you quickly deploy and manage Kubernetes, to scale and run applications while maintaining overall solid security.

This service started life as Azure Container Services (ACS) and supported Docker Swarm and Mesos/Mesosphere DC/OS at release to manage orchestrations. These original ACS workloads are still supported in Azure, but Kubernetes support was added.

It quickly became so popular that Microsoft changed the acronym for Azure Container Services to AKS and later changed the name of the service to Azure Kubernetes Service (also AKS).

## [Azure Container Registry \(ACR\)](#)

This service lets you store and manage container images in a central registry. It provides you with a Docker private registry as a first-class Azure resource.

All container deployments, including DC/OS, Docker Swarm, and Kubernetes, are supported. The registry is integrated with other Azure services such as the App Service, Batch, Service Fabric, and others.

Importantly, it allows your DevOps team to manage the configuration of apps without being tied to the configuration of the target-hosting environment.

### [Azure Container Apps](#)

Azure Container Apps allows you to build and deploy modern apps and microservices using serverless containers. It deploys containerized apps without managing complex infrastructure.

You can write code using your preferred programming language or framework and build microservices with full support for [Distributed Application Runtime \(Dapr\)](#). Scale dynamically based on HTTP traffic or events powered by [Kubernetes Event-Driven Autoscaling \(KEDA\)](#).

### [Azure App Service](#)

Azure Web Apps provides a managed service for both Windows and Linux-based web applications and provides the ability to deploy and run containerized applications for both platforms. It provides autoscaling and load balancing options and is easy to integrate with Azure DevOps.

## **Deploy Docker containers to Azure App Service web apps**

Completed 100 XP

- 30 minutes

**Estimated time:** 30 minutes.

**Lab files:** none.

### **Scenario**

In this lab, you will learn how to use an Azure DevOps CI/CD pipeline to build a custom Docker image, push it to Azure Container Registry, and deploy it as a container to Azure App Service.

## Objectives

After completing this lab, you'll be able to:

- Build a custom Docker image by using a Microsoft hosted Linux agent.
- Push an image to Azure Container Registry.
- Deploy a Docker image as a container to Azure App Service by using Azure DevOps.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft or Microsoft Entra account with the Contributor or the Owner role in the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](#) and [View and assign administrator roles in Microsoft Entra ID](#).

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Manage the service connection.
- Exercise 2: Import and run the CI pipeline.
- Exercise 3: Import and run the CD pipeline.
- Exercise 4: Remove the Azure lab resources

# Learn continuous integration with GitHub Actions

## Describe continuous integration with actions

Completed 100 XP

- 3 minutes

It's an example of a basic continuous integration workflow created by using actions:

```
YAMLCopy
name: dotnet Build

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [10.x]
    steps:
      - uses: actions/checkout@main
      - uses: actions/setup-dotnet@v1
        with:
          dotnet-version: '3.1.x'
      - run: dotnet build awesomeproject
```

- **On:** Specifies what will occur when code is pushed.
- **Jobs:** There's a single job called **build**.
- **Strategy:** It's being used to specify the Node.js version.
- **Steps:** Are doing a checkout of the code and setting up dotnet.
- **Run:** Is building the code.

## Examine environment variables

Completed 100 XP

- 3 minutes

When using Actions to create CI or CD workflows, you'll typically need to pass variable values to the actions. It's done by using Environment Variables.

## Built-in environment variables

GitHub provides a series of built-in environment variables. It all has a GITHUB\_ prefix.

### Note

Setting that prefix for your variables will result in an error.

Examples of built-in environment variables are:

**GITHUB\_WORKFLOW** is the name of the workflow.

**GITHUB\_ACTION** is the unique identifier for the action.

**GITHUB\_REPOSITORY** is the name of the repository (but also includes the name of the owner in owner/repo format)

## Using variables in workflows

Variables are set in the YAML workflow files. They're passed to the actions that are in the step.

YAMLCopy

```
jobs:  
  verify-connection:  
    steps:  
      - name: Verify Connection to SQL Server  
      - run: node testconnection.js  
    env:  
      PROJECT_SERVER: PH202323V  
      PROJECT_DATABASE: HAMaster
```

For more information on environment variables, including a list of built-in environment variables, see [Environment variables](#).

# Share artifacts between jobs

Completed 100 XP

- 4 minutes

When using Actions to create CI or CD workflows, you'll often need to pass artifacts created by one job to another.

The most common ways to do it are by using the **upload-artifact** and **download-artifact** actions.

## Upload-artifact

This action can upload one or more files from your workflow to be shared between jobs.

You can upload a specific file:

YAMLCopy

```
- uses: actions/upload-artifact
  with:
    name: harness-build-log
    path: bin/output/logs/harness.log
```

You can upload an entire folder:

YAMLCopy

```
- uses: actions/upload-artifact
  with:
    name: harness-build-logs
    path: bin/output/logs/
```

You can use wildcards:

YAMLCopy

```
- uses: actions/upload-artifact
  with:
    name: harness-build-logs
    path: bin/output/logs/harness[ab]?/*
```

You can specify multiple paths:

YAMLCopy

```
- uses: actions/upload-artifact
  with:
    name: harness-build-logs
    path: |
      bin/output/logs/harness.log
      bin/output/logs/harnessbuild.txt
```

For more information on this action, see [upload-artifact](#).

## Download-artifact

There's a corresponding action for downloading (or retrieving) artifacts.

YAMLCopy

```
- uses: actions/download-artifact
  with:
    name: harness-build-log
```

If no path is specified, it's downloaded to the current directory.

For more information on this action, see [download-artifact](#).

## Artifact retention

A default retention period can be set for the repository, organization, or enterprise.

You can set a custom retention period when uploading, but it can't exceed the defaults for the repository, organization, or enterprise.

YAMLCopy

```
- uses: actions/upload-artifact
  with:
    name: harness-build-log
```

```
path: bin/output/logs/harness.log  
retention-days: 12
```

## Deleting artifacts

You can delete artifacts directly in the GitHub UI.

For details, you can see: [Removing workflow artifacts](#).

## Examine Workflow badges

Completed 100 XP

- 3 minutes

Badges can be used to show the status of a workflow within a repository.

They show if a workflow is currently passing or failing. While they can appear in several locations, they typically get added to the README.md file for the repository.

Badges are added by using URLs. The URLs are formed as follows:

[https://github.com/<OWNER>/<REPOSITORY>/actions/workflows/<WORKFLOW\\_FILE>/badge.svg](https://github.com/<OWNER>/<REPOSITORY>/actions/workflows/<WORKFLOW_FILE>/badge.svg)

Where:

- AAAAA is the account name.
- RRRRR is the repository name.
- WWWW is the workflow name.

They usually indicate the status of the default branch but can be branch-specific. You do this by adding a URL query parameter:

?branch=BBBBB

where:

- BBBB is the branch name.

For more information, see: [Adding a workflow status badge](#).

## Describe best practices for creating actions

Completed 100 XP

- 2 minutes

It's essential to follow best practices when creating actions:

- Create chainable actions. Don't create large monolithic actions. Instead, create smaller functional actions that can be chained together.
- Version your actions like other code. Others might take dependencies on various versions of your actions. Allow them to specify versions.
- Provide the **latest** label. If others are happy to use the latest version of your action, make sure you provide the **latest** label that they can specify to get it.
- Add appropriate documentation. As with other codes, documentation helps others use your actions and can help avoid surprises about how they function.
- Add details **action.yml** metadata. At the root of your action, you'll have an **action.yml** file. Ensure it has been populated with author, icon, expected inputs, and outputs.
- Consider contributing to the marketplace. It's easier for us to work with actions when we all contribute to the marketplace. Help to avoid people needing to relearn the same issues endlessly.

## Mark releases with Git tags

Completed 100 XP

- 2 minutes

Releases are software iterations that can be packed for release.

In Git, releases are based on Git tags. These tags mark a point in the history of the repository. Tags are commonly assigned as releases are created.

Often these tags will contain version numbers, but they can have other values.

Tags can then be viewed in the history of a repository.

For more information on tags and releases, see: [About releases](#).

## Create encrypted secrets

Completed 100 XP

- 3 minutes

Actions often can use secrets within pipelines. Common examples are passwords or keys.

In GitHub actions, it's called **Secrets**.

## Secrets

Secrets are similar to environment variables but encrypted. They can be created at two levels:

- Repository
- Organization

If secrets are created at the organization level, access policies can limit the repositories that can use them.

## Creating secrets for a repository

To create secrets for a repository, you must be the repository's owner. From the repository **Settings**, choose **Secrets**, then **New Secret**.

For more information on creating secrets, see [Encrypted secrets](#).

## Use secrets in a workflow

Completed 100 XP

- 3 minutes

Secrets aren't passed automatically to the runners when workflows are executed.

Instead, when you include an action that requires access to a secret, you use the **secrets** context to provide it.

YAMLCopy

steps:

```
- name: Test Database Connectivity
  with:
    db_username: ${{ secrets.DBUserName }}
    db_password: ${{ secrets.DBPassword }}
```

## Command-line secrets

Secrets shouldn't be passed directly as command-line arguments as they may be visible to others. Instead, treat them like environment variables:

YAMLCopy

steps:

```
- shell: pwsh
  env:
    DB_PASSWORD: ${{ secrets.DBPassword }}
  run:
    db_test "$env:DB_PASSWORD"
```

## Limitations

Workflows can use up to 100 secrets, and they're limited to 64 KB in size.

For more information on creating secrets, see [Encrypted secrets](#).

# Implement GitHub Actions for CI/CD

Completed 100 XP

- 40 minutes

**Estimated time:** 40 minutes.

**Lab files:** none.

## Scenario

In this lab, you'll learn how to implement a GitHub Action workflow that deploys an Azure web app.

## Objectives

After completing this lab, you'll be able to:

- Implement a GitHub Action workflow for CI/CD.
- Explain the basic characteristics of GitHub Action workflows.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft or Microsoft Entra account with the Contributor or the Owner role in the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](#).
- If you don't already have a GitHub account that you can use for this lab, follow the instructions available at [Signing up for a new GitHub account](#) to create one.

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Import eShopOnWeb to your GitHub Repository.
- Exercise 1: Setup your GitHub Repository and Azure access.
- Exercise 2: Remove the Azure lab resources.

# Introduction to GitHub Actions

## What are Actions?

Completed 100 XP

- 1 minute

Actions are the mechanism used to provide workflow automation within the GitHub environment.

They're often used to build continuous integration (CI) and continuous deployment (CD) solutions.

However, they can be used for a wide variety of tasks:

- Automated testing.
- Automatically responding to new issues, mentions.
- Triggering code reviews.
- Handling pull requests.
- Branch management.

They're defined in YAML and stay within GitHub repositories.

Actions are executed on "runners," either hosted by GitHub or self-hosted.

Contributed actions can be found in the GitHub Marketplace. See [Marketplace Actions](#).

## Explore Actions flow

Completed 100 XP

- 1 minute

GitHub tracks events that occur. Events can trigger the start of workflows.

Workflows can also start on cron-based schedules and can be triggered by events outside of GitHub.

They can be manually triggered.

Workflows are the unit of automation. They contain Jobs.

Jobs use Actions to get work done.

## Understand workflows

Completed 100 XP

- 2 minutes

Workflows define the automation required. It details the events that should trigger the workflow.

Also, define the jobs that should run when the workflow is triggered.

The job defines the location in which the actions will run, like which runner to use.

Workflows are written in YAML and live within a GitHub repository at the place **.github/workflows**.

Example workflow:

YAMLCopy

```
# .github/workflows/build.yml
name: Node Build.

on: [push]

jobs:
  mainbuild:

    runs-on: ${{ matrix.os }}

  strategy:
    matrix:
      node-version: [12.x]
      os: [windows-latest]

  steps:
```

```
- uses: actions/checkout@v1
- name: Run node.js on latest Windows.
  uses: actions/setup-node@v1
  with:
    node-version: ${{ matrix.node-version }}

- name: Install NPM and build.
  run: |
    npm ci
    npm run build
```

You can find a set of starter workflows here: [Starter Workflows](#).

You can see the allowable syntax for workflows here: [Workflow syntax for GitHub Actions](#).

## Describe standard workflow syntax elements

Completed 100 XP

- 1 minute

Workflows include several standard syntax elements.

- **Name:** is the name of the workflow. It's optional but is highly recommended. It appears in several places within the GitHub UI.
- **On:** is the event or list of events that will trigger the workflow.
- **Jobs:** is the list of jobs to be executed. Workflows can contain one or more jobs.
- **Runs-on:** tells Actions which runner to use.
- **Steps:** It's the list of steps for the job. Steps within a job execute on the same runner.
- **Uses:** tells Actions, which predefined action needs to be retrieved. For example, you might have an action that installs node.js.
- **Run:** tells the job to execute a command on the runner. For example, you might execute an NPM command.

You can see the allowable syntax for workflows here: [Workflow syntax for GitHub Actions](#).

# Explore events

Completed 100 XP

- 2 minutes

Events are implemented by the **on** clause in a workflow definition.

There are several types of events that can trigger workflows.

## Scheduled events

With this type of trigger, a cron schedule needs to be provided.

YAMLCopy

```
on:  
  schedule:  
    - cron: '0 8-17 * * 1-5'
```

Cron schedules are based on five values:

- Minute (0 - 59)
- Hour (0 - 23)
- Day of the month (1 - 31)
- Month (1 - 12)
- Day of the week (0 - 6)

Aliases for the months are JAN-DEC and for days of the week are SUN-SAT.

A wild card means any. (\*) is a special value in YAML, so the cron string will need to be quoted)

So, in the example above, the schedule would be 8 AM - 5 PM Monday to Friday.

## Code events

Code events will trigger most actions. It occurs when an event of interest occurs in the repository.

YAMLCopy

```
on:  
  pull_request
```

The above event would fire when a pull request occurs.

YAMLCopy

```
on:  
  [push, pull_request]
```

The above event would fire when either a push or a pull request occurs.

YAMLCopy

```
on:  
  pull_request:  
    branches:  
      - develop
```

The event shows how to be specific about the section of the code that is relevant.

In this case, it will fire when a pull request is made in the develop branch.

## Manual events

There's a unique event that is used to trigger workflow runs manually. You should use the **workflow\_dispatch** event.

Your workflow must be in the default branch for the repository.

## Webhook events

Workflows can be executed when a GitHub webhook is called.

YAMLCopy

```
on:  
  gollum
```

This event would fire when someone updates (or first creates) a Wiki page.

## External events

Events can be on **repository\_dispatch**. That allows events to fire from external systems.

For more information on events, see [Events that trigger workflows](#).

## Explore jobs

Completed 100 XP

- 2 minutes

Workflows contain one or more jobs. A job is a set of steps that will be run in order on a runner.

Steps within a job execute on the same runner and share the same filesystem.

The logs produced by jobs are searchable, and artifacts produced can be saved.

## Jobs with dependencies

By default, if a workflow contains multiple jobs, they run in parallel.

YAMLCopy

```
jobs:  
  startup:  
    runs-on: ubuntu-latest  
    steps:  
      - run: ./setup_server_configuration.sh  
  build:  
    steps:  
      - run: ./build_new_server.sh
```

Sometimes you might need one job to wait for another job to complete.

You can do that by defining dependencies between the jobs.

YAMLCopy

```
jobs:
```

```
startup:  
  runs-on: ubuntu-latest  
  steps:  
    - run: ./setup_server_configuration.sh  
build:  
  needs: startup  
  steps:  
    - run: ./build_new_server.sh
```

## Note

If the startup job in the example above fails, the build job won't execute.

For more information on job dependencies, see the section **Creating Dependent Jobs** at [Managing complex workflows](#).

# Explore runners

Completed 100 XP

- 2 minutes

When you execute jobs, the steps execute on a Runner.

The steps can be the execution of a shell script or the execution of a predefined Action.

GitHub provides several hosted runners to avoid you needing to spin up your infrastructure to run actions.

Now, the maximum duration of a job is 6 hours, and for a workflow is 72 hours.

For JavaScript code, you have implementations of node.js on:

- Windows
- macOS
- Linux

If you need to use other languages, a Docker container can be used. Now, the Docker container support is only Linux-based.

These options allow you to write in whatever language you prefer.

JavaScript actions will be faster (no container needs to be used) and more versatile runtime.

The GitHub UI is also better for working with JavaScript actions.

## Self-hosted runners

If you need different configurations to the ones provided, you can create a self-hosted runner.

GitHub has published the source code for self-hosted runners as open-source, and you can find it here: <https://github.com/actions/runner>.

It allows you to customize the runner completely. However, you then need to maintain (patch, upgrade) the runner system.

Self-hosted runners can be added at different levels within an enterprise:

- Repository-level (single repository).
- Organizational-level (multiple repositories in an organization).
- Enterprise-level (multiple organizations across an enterprise).

GitHub strongly recommends that you don't use self-hosted runners in public repos.

Doing it would be a significant security risk, as you would allow someone (potentially) to run code on your runner within your network.

For more information on self-hosted runners, see: [About self-hosted runners](#).

## Examine release and test an action

Completed 100 XP

- 4 minutes

Actions will often produce console output. You don't need to connect directly to the runners to retrieve that output.

The console output from actions is available directly from within the GitHub UI.

Select **Actions** on the top repository menu to see a list of executed workflows to see the output.

Next, click on the job's name to see the steps' output.

Console output can help debug. If it isn't sufficient, you can also enable more logging. See: [Enabling debug logging](#).

## Release Management for Actions

While you might be happy to retrieve the latest version of the action, there are many situations where you might want a specific version of the action.

You can request a specific release of action in several ways:

### Tags

Tags allow you to specify the precise versions that you want to work.

YAMLCopy

```
steps:  
  -uses: actions/install-timer@v2.0.1
```

### SHA-based hashes

You can specify a requested SHA-based hash for an action. It ensures that the action hasn't changed. However, the downside to this is that you also won't receive updates to the action automatically either.

YAMLCopy

```
steps:  
  -uses: actions/install-timer@327239021f7cc39fe7327647b213799853a9eb98
```

## Branches

A common way to request actions is to refer to the branch you want to work with. You'll then get the latest version from that branch. That means you'll benefit from updates, but it also increases the chance of code-breaking.

YAMLCopy

```
steps:  
  -uses: actions/install-timer@develop
```

## Test an Action

GitHub offers several learning tools for actions.

[GitHub Actions: hello-world](#)

You'll see a basic example of how to:

- Organize and identify workflow files.
- Add executable scripts.
- Create workflow and action blocks.
- Trigger workflows.
- Discover workflow logs.

# Integrate with Azure Pipelines

## Describe the anatomy of a pipeline

Completed 100 XP

- 6 minutes

Azure Pipelines can automatically build and validate every pull request and commit to your Azure Repos Git repository.

Azure Pipelines can be used with Azure DevOps public projects and Azure DevOps private projects.

In future training sections, we'll also learn how to use Azure Repos with external code repositories such as GitHub.

Let's start by creating a hello world YAML Pipeline.

## Hello world

Start slowly and create a pipeline that echoes "Hello world!" to the console. No technical course is complete without a hello world example.

YAMLCopy

```
name: 1.0$(Rev:.r)

# simplified trigger (implied branch)
trigger:
- main

# equivalents trigger
# trigger:
# branches:
#   include:
#     - main

variables:
  name: John

pool:
  vmImage: ubuntu-latest

jobs:
```

```
- job: helloworld
  steps:
    - checkout: self
    - script: echo "Hello, $(name)"
```

Most pipelines will have these components:

- Name – though often it's skipped (if it's skipped, a date-based name is generated automatically).
- Trigger – more on triggers later, but without an explicit trigger. There's an implicit "trigger on every commit to any path from any branch in this repo."
- Variables – "Inline" variables (more on other types of variables later).
- Job – every pipeline must have at least one job.
- Pool – you configure which pool (queue) the job must run on.
- Checkout – the "checkout: self" tells the job which repository (or repositories if there are multiple checkouts) to check out for this job.
- Steps – the actual tasks that need to be executed: in this case, a "script" task (the script is an alias) that can run inline scripts.

## Name

The variable name is a bit misleading since the name is in the build number format. You'll get an integer number if you don't explicitly set a name format. A monotonically increasing number of runs triggered off this pipeline, starting at 1. This number is stored in Azure DevOps. You can make use of this number by referencing \$(Rev).

To make a date-based number, you can use the format \$(Date:yyyyMMdd) to get a build number like 20221003.

To get a semantic number like 1.0.x, you can use something like 1.0.\$(Rev:r).

## Triggers

If there's no explicit triggers section, then it's implied that any commit to any path in any branch will trigger this pipeline to run.

However, you can be more precise by using filters such as branches or paths.

Let's consider this trigger:

YAMLCopy

```
trigger:  
  branches:  
    include:  
  
    - main
```

This trigger is configured to queue the pipeline only when a commit to the main branch exists. What about triggering for any branch except the main? You guessed it: use exclude instead of include:

YAMLCopy

```
trigger:  
  branches:  
    exclude:  
  
    - main
```

## Tip

You can get the name of the branch from the variables Build.SourceBranch (for the full name like refs/heads/main) or Build.SourceBranchName (for the short name like main).

What about a trigger for any branch with a name that starts with topic/ and only if the change is in the webapp folder?

Copy

```
trigger:  
  branches:  
    include:  
  
    - feature/*  
paths:  
  include:  
  
  - webapp/**
```

You can mix includes and excludes if you need to. You can also filter on tags.

## Tip

Don't forget one overlooked trigger: none. You can use none if you never want your pipeline to trigger automatically. It's helpful if you're going to create a pipeline that is only manually triggered.

There are other triggers for other events, such as:

- Pull Requests (PRs) can also filter branches and paths.
- Schedules allow you to specify cron expressions for scheduling pipeline runs.
- Pipelines will enable you to trigger pipelines when other pipelines are complete, allowing pipeline chaining.

You can find all the documentation on triggers [here](#).

## Jobs

A job is a set of steps an agent executes in a queue (or pool). Jobs are atomic – they're performed wholly on a single agent. You can configure the same job to run on multiple agents simultaneously, but even in this case, the entire set of steps in the job is run on every agent. You'll need two jobs if you need some steps to run on one agent and some on another.

A job has the following attributes besides its name:

- `displayName` – a friendly name.
- `dependsOn` - a way to specify dependencies and ordering of multiple jobs.
- `condition` – a binary expression: if it evaluates to true, the job runs; if false, the job is skipped.
- `strategy` - used to control how jobs are parallelized.
- `continueOnError` - specify if the rest of the pipeline should continue if this job fails.
- `pool` – the pool name (queue) to run this job on.
- `workspace` - managing the source workspace.
- `container` - for specifying a container image to execute the job later.
- `variables` – variables scoped to this job.
- `steps` – the set of steps to execute.
- `timeoutInMinutes` and `cancelTimeoutInMinutes` for controlling timeouts.
- `services` - sidecar services that you can spin up.

## Dependencies

When you define multiple stages in a pipeline, by default, they run sequentially in the order in which you define them in the YAML file. The exception to this is when you add dependencies. With dependencies, stages run in the order of the dependsOn requirements.

Pipelines must contain at least one stage with no dependencies.

Let's look at a few examples. Consider this pipeline:

```
YAMLCopy  
jobs:  
- job: A  
  steps:  
    # steps omitted for brevity  
  
- job: B  
  steps:  
    # steps omitted for brevity
```

Because no dependsOn was specified, the jobs will run sequentially: first A and then B.

To have both jobs run in parallel, we add dependsOn: [] to job B:

```
YAMLCopy  
jobs:  
- job: A  
  steps:  
    # steps omitted for brevity  
  
- job: B  
  dependsOn: [] # This removes the implicit dependency on the previous stage and  
  causes this to run in parallel.  
  steps:  
    # steps omitted for brevity
```

If we want to fan out and fan in, we can do that too:

```
YAMLCopy  
jobs:  
- job: A
```

```
steps:  
  - script: echo' job A.'  
  
  - job: B  
    dependsOn: A  
    steps:  
      - script: echo' job B.'  
  
  - job: C  
    dependsOn: A  
    steps:  
      - script: echo' job C.'  
  
  - job: D  
    dependsOn:  
      - B  
      - C  
    steps:  
      - script: echo' job D.'  
  
  - job: E  
    dependsOn:  
      - B  
      - D  
    steps:  
      - script: echo' job E.'
```

## Checkout

Classic builds implicitly checkout any repository artifacts, but pipelines require you to be more explicit using the `checkout` keyword:

- Jobs check out the repo they're contained in automatically unless you specify `checkout: none`.

- Deployment jobs don't automatically check out the repo, so you'll need to specify checkout: self for deployment jobs if you want access to the YAML file's repo.

## Download

Downloading artifacts requires you to use the download keyword. Downloads also work the opposite way for jobs and deployment jobs:

- Jobs don't download anything unless you explicitly define a download.
- Deployment jobs implicitly do a download: current, which downloads any pipeline artifacts created in the existing pipeline. To prevent it, you must specify download: none.

## Resources

What if your job requires source code in another repository? You'll need to use resources. Resources let you reference:

- other repositories
- pipelines
- builds (classic builds)
- containers (for container jobs)
- packages

To reference code in another repo, specify that repo in the resources section and then reference it via its alias in the checkout step:

YAMLCopy

```
resources:  
  repositories:  
  
    - repository: appcode  
      type: git  
      name: otherRepo  
  
steps:  
  
  - checkout: appcode
```

## Steps are Tasks

Steps are the actual "things" that execute in the order specified in the job.

Each step is a task: out-of-the-box (OOB) tasks come with Azure DevOps. Many have aliases and tasks installed on your Azure DevOps organization via the marketplace.

Creating custom tasks is beyond the scope of this chapter, but you can see how to make your custom tasks [here](#).

## Variables

It would be tough to achieve any sophistication in your pipelines without variables. Though this classification is partly mine, several types of variables exist, and pipelines don't distinguish between these types. However, I've found it helpful to categorize pipeline variables to help teams understand nuances when dealing with them.

Every variable is a key: value pair. The key is the variable's name, and it has a value.

To dereference a variable, wrap the key in `$()`. Let's consider this example:

YAMLCopy

```
variables:  
  name: John  
steps:  
- script: echo "Hello, $(name)!"
```

It will write Hello, John! To the log.

## Understand the pipeline structure

Completed 100 XP

- 4 minutes

A pipeline is one or more stages that describe a CI/CD process.

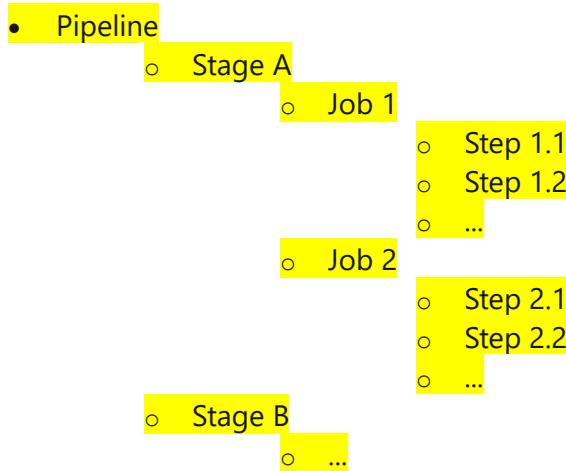
Stages are the primary divisions in a pipeline. The stages "Build this app," "Run these tests," and "Deploy to preproduction" are good examples.

A stage is one or more jobs, units of work assignable to the same machine.

You can arrange both stages and jobs into dependency graphs. Examples include "Run this stage before that one" and "This job depends on the output of that job."

A job is a linear series of steps. Steps can be tasks, scripts, or references to external templates.

This hierarchy is reflected in the structure of a YAML file like:



Simple pipelines don't require all these levels. For example, you can omit the containers for stages and jobs in a single job build because there are only steps.

Because many options shown in this article aren't required and have reasonable defaults, your YAML definitions are unlikely to include all of them.

## Pipeline

The schema for a pipeline:

YAMLCopy

```
name: string # build numbering format
resources:
  pipelines: [ pipelineResource ]
  containers: [ containerResource ]
  repositories: [ repositoryResource ]
variables: # several syntaxes
trigger: trigger
pr: pr
stages: [ stage | templateReference ]
```

If you have a single-stage, you can omit the stages keyword and directly specify the jobs keyword:

YAMLCopy

```
# ... other pipeline-level keywords
jobs: [ job | templateReference ]
```

If you've a single-stage and a single job, you can omit the stages and jobs keywords and directly specify the steps keyword:

YAMLCopy

```
# ... other pipeline-level keywords
steps: [ script | bash | pwsh | powershell | checkout | task | templateReference ]
```

## Stage

A stage is a collection of related jobs. By default, stages run sequentially. Each stage starts only after the preceding stage is complete.

Use approval checks to control when a stage should run manually. These checks are commonly used to control deployments to production environments.

Checks are a mechanism available to the resource owner. They control when a stage in a pipeline consumes a resource.

As an owner of a resource like an environment, you can define checks required before a stage that consumes the resource can start.

This example runs three stages, one after another. The middle stage runs two jobs in parallel.

YAMLCopy

```
stages:
- stage: Build
  jobs:
    - job: BuildJob
      steps:
        - script: echo Building!
```

```

- stage: Test
dependsOn: Build
jobs:
  - job: TestOnWindows
    steps:
      - script: echo Testing on Windows!
  - job: TestOnLinux
    steps:
      - script: echo Testing on Linux!
- stage: Deploy
dependsOn: Test
jobs:
  - job: Deploy
    steps:
      - script: echo Deploying the code!

```

## Job

A job is a collection of steps run by an agent or on a server. Jobs can run conditionally and might depend on previous jobs.

YAMLCopy

```

jobs:
  - job: MyJob
    displayName: My First Job
    continueOnError: true
    workspace:
      clean: outputs
    steps:
      - script: echo My first job

```

## Deployment strategies

Deployment strategies allow you to use specific techniques to deliver updates when deploying your application.

Techniques examples:

- 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.

## RunOnce

`runOnce` is the most straightforward deployment strategy in all the presented lifecycle hooks.

## YAMLCopy

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

For details and examples, see [Deployment jobs](#).

## 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`.

## YAMLCopy

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

For details and examples, see [Deployment jobs](#).

## Canary

Using this strategy, you can first roll out the changes to a small subset of servers. The canary deployment strategy is an advanced deployment strategy that helps mitigate the risk of rolling out new versions of applications.

As you gain more confidence in the new version, you can release it to more servers in your infrastructure and route more traffic to it.

## YAMLCopy

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

```

...
routeTraffic:
  pool: [ server | pool ]
  steps:
  ...
postRouteTraffic:
  pool: [ server | pool ]
  steps:
  ...
on:
  failure:
    pool: [ server | pool ]
    steps:
  ...
  success:
    pool: [ server | pool ]
    steps:
  ...

```

For details and examples, see [Deployment jobs](#).

## Lifecycle hooks

You can achieve the deployment strategies technique by using lifecycle hooks. Depending on the pool attribute, each resolves into an agent or [server job](#).

Lifecycle hooks inherit the pool specified by the deployment job. Deployment jobs use the `$(Pipeline.Workspace)` system variable.

Available lifecycle hooks:

- **preDeploy:** Used to run steps that initialize resources before application deployment starts.
- **deploy:** Used to run steps that deploy your application. Download artifact task will be auto-injected only in the deploy hook for deployment jobs. To stop downloading artifacts, use `- download: none` or choose specific artifacts to download by specifying [Download Pipeline Artifact task](#).
- **routeTraffic:** Used to run steps that serve the traffic to the updated version.
- **postRouteTraffic:** Used to run the steps after the traffic is routed. Typically, these tasks monitor the health of the updated version for a defined interval.
- **on: failure** or **on: success:** Used to run steps for rollback actions or clean-up.

## Steps

A step is a linear sequence of operations that make up a job. Each step runs its process on an agent and accesses the pipeline workspace on a local hard drive.

This behavior means environment variables aren't preserved between steps, but file system changes are.

YAMLCopy

steps:

```
- script: echo This run in the default shell on any machine
- bash: |
  echo This multiline script always runs in Bash.
  echo Even on Windows machines!

- pwsh: |
  Write-Host "This multiline script always runs in PowerShell Core."
  Write-Host "Even on non-Windows machines!"
```

## Tasks

Tasks are the building blocks of a pipeline. There's a catalog of tasks available to choose from.

YAMLCopy

steps:

```
- task: VSBuild@1
  displayName: Build
  timeoutInMinutes: 120
  inputs:
    solution: '**\*.sln'
```

## Detail templates

Completed 100 XP

- 3 minutes

## Template references

You can export reusable sections of your pipeline to a separate file. These individual files are known as templates.

Azure Pipelines supports four types of templates:

- Stage
- Job
- Step
- Variable

You can also use templates to control what is allowed in a pipeline and define how parameters can be used.

- Parameter

Templates themselves can include other templates. Azure Pipelines supports 50 individual template files in a single pipeline.

## Stage templates

You can define a set of stages in one file and use it multiple times in other files.

In this example, a stage is repeated twice for two testing regimes. The stage itself is specified only once.

```
YAMLCopy
# File: stages/test.yml

parameters:
  name: ''
  testFile: ''

stages:
- stage: Test_${{ parameters.name }}
  jobs:
    - job: ${{ parameters.name }}_Windows
      pool:
        vmImage: windows-latest
      steps:
        - script: npm install
        - script: npm test -- --file=${{ parameters.testFile }}

    - job: ${{ parameters.name }}_Mac
      pool:
        vmImage: macOS-latest
      steps:
```

```
- script: npm install
- script: npm test -- --file=${{ parameters.testFile }}
```

## Templated pipeline

### YAMLCopy

```
# File: azure-pipelines.yml

stages:

- template: stages/test.yml # Template reference
  parameters:
    name: Mini
    testFile: tests/miniSuite.js

- template: stages/test.yml # Template reference
  parameters:
    name: Full
    testFile: tests/fullSuite.js
```

## Job templates

You can define a set of jobs in one file and use it multiple times in other files.

In this example, a single job is repeated on three platforms. The job itself is specified only once.

### YAMLCopy

```
# 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
```

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

jobs:

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

- template: jobs/build.yml # Template reference
  parameters:
    name: Linux
    pool:
      vmImage: 'ubuntu-latest'

- template: jobs/build.yml # Template reference
  parameters:
    name: Windows
    pool:
      vmImage: 'windows-latest'
    sign: true # Extra step on Windows only
```

## Step templates

You can define a set of steps in one file and use it multiple times in another.

```
YAMLCopy
# File: steps/build.yml

steps:

- script: npm install
- script: npm test

YAMLCopy
# File: azure-pipelines.yml

jobs:

- job: macOS
  pool:
    vmImage: 'macOS-latest'
  steps:

    - template: steps/build.yml # Template reference
```

```

- job: Linux
  pool:
    vmImage: 'ubuntu-latest'
  steps:
    - template: steps/build.yml # Template reference

- job: Windows
  pool:
    vmImage: 'windows-latest'
  steps:
    - template: steps/build.yml # Template reference
    - script: sign           # Extra step on Windows only

```

## Variable templates

You can define a set of variables in one file and use it multiple times in other files.

In this example, a set of variables is repeated across multiple pipelines. The variables are specified only once.

YAMLCopy

```

# File: variables/build.yml
variables:

- name: vmImage
  value: windows-latest

- name: arch
  value: x64

- name: config
  value: debug

```

YAMLCopy

```

# File: component-x-pipeline.yml
variables:

- template: variables/build.yml # Template reference
pool:
  vmImage: ${{ variables.vmImage }}
steps:

- script: build x ${{ variables.arch }} ${{ variables.config }}

```

```
YAMLCopy
# File: component-y-pipeline.yml
variables:

- template: variables/build.yml # Template reference
pool:
  vmImage: ${{ variables.vmImage }}
steps:

- script: build y ${{ variables.arch }} ${{ variables.config }}
```

## Explore YAML resources

Completed 100 XP

- 2 minutes

Resources in YAML represent sources of pipelines, repositories, and containers. For more information on Resources, [see here](#).

## General schema

YAMLCopy

```
resources:
  pipelines: [ pipeline ]
  repositories: [ repository ]
  containers: [ container ]
```

## Pipeline resource

If you have an Azure pipeline that produces artifacts, your pipeline can consume the artifacts by using the pipeline keyword to define a pipeline resource.

YAMLCopy

```
resources:
  pipelines:
    - pipeline: MyAppA
      source: MyCIPipelineA

    - pipeline: MyAppB
      source: MyCIPipelineB
      trigger: true
```

```
- pipeline: MyAppC
project: DevOpsProject
source: MyCIPipelineC
branch: releases/M159
version: 20190718.2
trigger:
  branches:
    include:
      - master
      - releases/*
    exclude:
      - users/*
```

## Container resource

Container jobs let you isolate your tools and dependencies inside a container. The agent launches an instance of your specified container then runs steps inside it. The container keyword lets you specify your container images.

Service containers run alongside a job to provide various dependencies like databases.

### YAMLCopy

```
resources:
  containers:

    - container: linux
      image: ubuntu:16.04

    - container: windows
      image: myprivate.azurecr.io/windowsservercore:1803
      endpoint: my_acr_connection

    - container: my_service
      image: my_service:tag
      ports:
        - 8080:80 # bind container port 80 to 8080 on the host machine
        - 6379 # bind container port 6379 to a random available port on the host machine
      volumes:
        - /src/dir:/dst/dir # mount /src/dir on the host into /dst/dir in the container
```

## Repository resource

Let the system know about the repository if:

- If your pipeline has templates in another repository.
- If you want to use multi-repo checkout with a repository that requires a service connection.

The repository keyword lets you specify an external repository.

YAMLCopy

```
resources:  
  repositories:  
  
    - repository: common  
      type: github  
      name: Contoso/CommonTools  
      endpoint: MyContosoServiceConnection
```

## Use multiple repositories in your pipeline

Completed 100 XP

- 3 minutes

You might have micro git repositories providing utilities used in multiple pipelines within your project. Pipelines often rely on various repositories.

You can have different repositories with sources, tools, scripts, or other items that you need to build your code. By using multiple checkout steps in your pipeline, you can fetch and check out other repositories to the one you use to store your YAML pipeline.

Previously Azure Pipelines hasn't offered support for using multiple code repositories in a single pipeline. Using artifacts or directly cloning other repositories via script within a pipeline, you can work around it. It leaves access management and security down to you.

Repositories are now first-class citizens within Azure Pipelines. It enables some exciting use cases, such as checking out specific repository parts and checking multiple repositories.

There's also a use case for not checking out any repository in the pipeline. It can be helpful in cases where you're setting up a pipeline to do a job that has no dependency on any repository.

# Specify multiple repositories

Repositories can be specified as a repository resource or in line with the checkout step. Supported repositories are Azure Repos Git, GitHub, and BitBucket Cloud.

The following combinations of checkout steps are supported.

- If there are no **checkout** steps, the default behavior is **checkout: self** is the first step.
- If there's a single **checkout: none** step, no repositories are synced or checked out.
- If there's a single **checkout: self** step, the current repository is checked out.
- If there's a single **checkout** step that isn't **self** or **none**, that repository is checked out instead of self.
- If there are multiple **checkout** steps, each named repository is checked out to a folder named after the repository. Unless a different path is specified in the **checkout** step, use **checkout: self** as one of the **checkout** steps.

## Repository resource - How to do it?

If your repository type requires a service connection or other extended resources field, you must use a repository resource.

Even if your repository type doesn't require a service connection, you may use a repository resource.

For example, you have a repository resource defined already for templates in a different repository.

In the following example, three repositories are declared as repository resources. The repositories and the current self-repository containing the pipeline YAML are checked out.

### YAMLCopy

```
resources:
  repositories:

  - repository: MyGitHubRepo # The name used to reference this repository in the
    checkout step.
    type: github
    endpoint: MyGitHubServiceConnection
    name: MyGitHubOrgOrUser/MyGitHubRepo

  - repository: MyBitBucketRepo
```

```

type: bitbucket
endpoint: MyBitBucketServiceConnection
name: MyBitBucketOrgOrUser/MyBitBucketRepo

- repository: MyAzureReposGitRepository
  type: git
  name: MyProject/MyAzureReposGitRepo

trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- checkout: self
- checkout: MyGitHubRepo
- checkout: MyBitBucketRepo
- checkout: MyAzureReposGitRepository

- script: dir $(Build.SourcesDirectory)

```

If the self-repository is named CurrentRepo, the script command produces the following output: CurrentRepo MyAzureReposGitRepo MyBitBucketRepo MyGitHubRepo.

In this example, the repositories' names are used for the folders because no path is specified in the checkout step.

## Inline - How to do it?

If your repository doesn't require a service connection, you can declare it according to your checkout step.

YAMLCopy

steps:

```
- checkout: git://MyProject/MyRepo # Azure Repos Git repository in the same organization
```

The default branch is checked out unless you choose a specific ref.

If you're using inline syntax, choose the ref by appending @ref. For example:

YAMLCopy

```
- checkout: git://MyProject/MyRepo@features/tools # checks out the features/tools branch
- checkout: git://MyProject/MyRepo@refs/heads/features/tools # also checks out the features/tools branch.
- checkout: git://MyProject/MyRepo@refs/tags/MyTag # checks out the commit referenced by MyTag.
```

## GitHub repository

Azure Pipelines can automatically build and validate every pull request and commit to your GitHub repository.

When creating your new pipeline, you can select a GitHub repository and then a YAML file in that repository (self repository). By default, this is the repository that your pipeline builds.

Azure Pipelines must be granted access to your repositories to trigger their builds and fetch their code during builds.

There are three authentication types for granting Azure Pipelines access to your GitHub repositories while creating a pipeline.

- GitHub App.
- OAuth.
- Personal access token (PAT).

You can create a continuous integration (CI) trigger to run a pipeline whenever you push an update to the specified branches or push selected tags.

YAML pipelines are configured by default with a CI trigger on all branches.

YAMLCopy

```
trigger:
  - main
  - releases/*
```

You can configure complex triggers that use **exclude** or **batch**.

YAMLCopy

```
# specific branches build
trigger:
  branches:
    include:
      - master
      - releases/*
  exclude:
    - releases/old*
```

Also, it's possible to configure pull request (PR) triggers to run whenever a pull request is opened with one of the specified target branches or when updates are made to such a pull request.

You can specify the target branches when validating your pull requests.

To validate pull requests that target main and releases/\* and start a new run the first time a new pull request is created, and after every update made to the pull request:

**YAMLCopy**

```
pr:
  - main
  - releases/*
```

You can specify the full name of the branch or a wildcard.

For more information and guidance about GitHub integration, see:

- [Build GitHub repositories](#).

# Implement a pipeline strategy

## Configure agent demands

Completed 100 XP

- 2 minutes

Not all agents are the same. We've seen that they can be based on different operating systems, but they can also install different dependencies.

To describe it, every agent has a set of capabilities configured as name-value pairs. The capabilities such as machine name and operating system type that are automatically discovered are referred to as **System capabilities**. The ones that you define are called **User-defined capabilities**.

There's a tab for Capabilities on the Agent Pools page (at the Organization level) **when you select an agent**.

You can use it to see the available capabilities for an agent and to configure user capabilities.

Opening a configured self-hosted agent, you can see the capabilities on that tab:

Jobs Capabilities

### User-defined capabilities



No user-defined capabilities

[Add a new capability](#)

### System capabilities

[Search by keyword](#)

| Name                 | Value   |
|----------------------|---|
| Agent.Name           |   |
| Agent.Version        | 2.202.0                                       |
| Agent.ComputerName   |   |
| Agent.HomeDirectory  | C:\DevOpsAgents\01                            |
| Agent.OS             | Windows_NT                                    |
| Agent.OSArchitecture | X64   |
| Agent.OSVersion      | 10.0.22000                                    |
| ALLUSERSPROFILE      | C:\ProgramData                                |
| APPDATA              | C:\WINDOWS\ServiceProfiles\NetworkService\... |

When you configure a build pipeline and the agent pool to use, you can specify specific demands that the agent must meet on the Options tab.

**Build job**  
Define build job authorization and timeout settings

Build job authorization scope ⓘ

Project collection

Build job timeout in minutes ⓘ

60

Build job cancel timeout in minutes ⓘ

5

**Demands**  
Specify which capabilities the agent must have to run this pipeline.

| Name              | Condition | Value |
|-------------------|-----------|-------|
| HasPaymentService | exists    | ▼     |

+ Add

In the build job image, the HasPaymentService is required in the collection of capabilities. And an **exists** condition, you can choose that a capability **equals** a specific value.

For more information, see [Capabilities](#).

## Implement multi-agent builds

Completed 100 XP

- 3 minutes

You can use multiple build agents to support multiple build machines. Either distribute the load, run builds in parallel, or use different agent capabilities.

As an example, components of an application might require different incompatible versions of a library or dependency.

A screenshot of the Azure Pipelines pipeline editor. At the top, there are tabs for Tasks, Variables, Triggers, Options, Retention, History, Save & queue, Discard, and Summary. Below the tabs, the pipeline is titled "Pipeline" and "Build pipeline". It contains a "Get sources" task (PartsUnlimited, master branch) and an "Agent job 1" task (Run on agent). To the right of the pipeline, a context menu is open with a red arrow pointing to the "Add an agent job" option. Other options in the menu include "Add an agentless job" and "Learn more about jobs".

## Multiple jobs in a pipeline

Adding multiple jobs to a pipeline lets you:

- Break your pipeline into sections that need different agent pools or self-hosted agents.
- Publish artifacts in one job and consume them in one or more subsequent jobs.
- Build faster by running multiple jobs in parallel.
- Enable conditional execution of tasks.

To add another agent job to an existing pipeline, click on the ellipsis and choose as shown in this image:

A screenshot of the Azure Pipelines pipeline editor, identical to the one above but with a different context menu. A red arrow points to the "Add an agent job" option in the menu, which is highlighted with a hand cursor icon. The menu also includes "Add an agentless job" and "Learn more about jobs".

## Parallel jobs

At the organization level, you can configure the number of parallel jobs that are made available.

**Organization Settings**

**Parallel jobs**

**Private projects**

| Category                              | Value  | Actions  |
|---------------------------------------|--|--|
| Microsoft-hosted                      | Free tier<br>1 parallel job up to 1800 mins/mo | <a href="#">View in-progress jobs</a> <a href="#">Purchase parallel jobs</a> |
| Currently 5/1800 minutes are consumed |  |  |
| Self-hosted                           | 2 Parallel jobs                                | <a href="#">View in-progress jobs</a>  |
| Free parallel jobs                    | 1  |  |
| Visual Studio Enterprise subscribers  | 1  |  |
| Monthly purchases                     | 0 <a href="#">Change</a>                       |  |

**Public projects** Free

| Category         | Value            | Actions                               |
|------------------|------------------|---------------------------------------|
| Microsoft-hosted | 10 Parallel jobs | <a href="#">View in-progress jobs</a> |
| Self-hosted      | 10 Parallel jobs | <a href="#">View in-progress jobs</a> |

**Retention and parallel jobs** 

**OAuth configurations**

The free tier allows for one parallel job of up to 1800 minutes per month. The self-hosted agents have higher levels.

**Note**

You can define a build as a collection of jobs rather than as a single job. Each job consumes one of these parallel jobs that run on an agent. If there aren't enough parallel jobs available for your organization, the jobs will be queued and run sequentially.

## Build Related Tooling

Azure DevOps can be integrated with a wide range of existing tooling used for builds or associated with builds.

Which build-related tools do you currently use?

What do you like or don't like about the tools?

## Explore source control types supported by Azure Pipelines

Completed 100 XP

- 1 minute

Azure Pipelines offers both YAML-based pipelines and the classic editor.

The table shows the repository types supported by both.

| Repository type          | Azure Pipelines (YAML) | Azure Pipelines (classic editor) |
|--------------------------|------------------------|----------------------------------|
| Azure Repos Git          | Yes                    | Yes                              |
| Azure Repos TFVC         | No                     | Yes                              |
| Bitbucket Cloud          | Yes                    | Yes                              |
| Other Git (generic)      | No                     | Yes                              |
| GitHub                   | Yes                    | Yes                              |
| GitHub Enterprise Server | Yes                    | Yes                              |
| Subversion               | No                     | Yes                              |

# Explore continuous integration

## Introduction to continuous integration

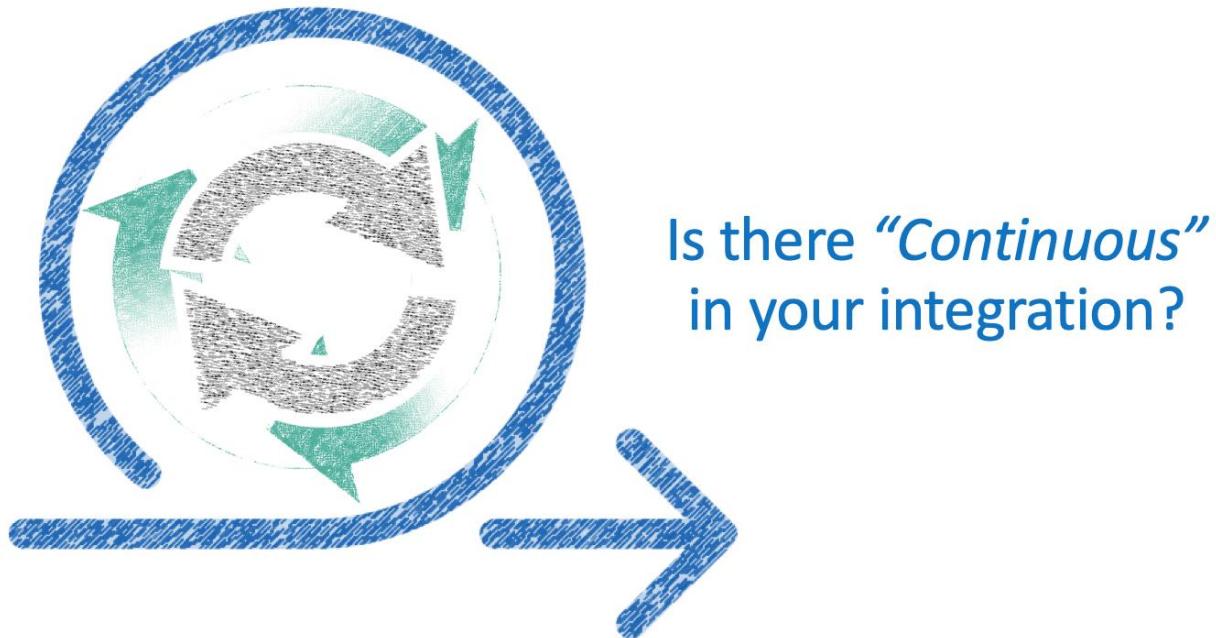
Completed 100 XP

- 3 minutes

Continuous integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control.

CI encourages developers to share their code and unit tests by merging their changes into a shared version control repository after every small task completion.

Committing code triggers an automated build system to grab the latest code from the shared repository and build, test, and validate the entire main branch (also known as the trunk or main).



The idea is to minimize the cost of integration by making it an early consideration.

Developers can discover conflicts at the boundaries between new and existing code early, while conflicts are still relatively easy to reconcile.

Once the conflict is resolved, work can continue with confidence that the new code honors the requirements of the existing codebase.

Integrating code frequently doesn't offer any guarantees about the quality of the new code or functionality.

In many organizations, integration is costly because manual processes ensure that the code meets standards, introduces bugs, and breaks existing functionality.

Frequent integration can create friction when the level of automation doesn't match the amount of quality assurance measures in place.

In practice, continuous integration relies on robust test suites and an automated system to run those tests to address this friction within the integration process.

When a developer merges code into the main repository, automated processes kick off a build of the new code.

Afterward, test suites are run against the new build to check whether any integration problems were introduced.

If either the build or the test phase fails, the team is alerted to work to fix the build.

The end goal of continuous integration is to make integration a simple, repeatable process part of the everyday development workflow to reduce integration costs and respond to early defects.

Working to make sure the system is robust, automated, and fast while cultivating a team culture that encourages frequent iteration and responsiveness to build issues is fundamental to the strategy's success.

## Learn the four pillars of continuous integration

Completed 100 XP

- 4 minutes

Continuous integration relies on four key elements for successful implementation: a Version Control System, Package Management System, Continuous Integration System, and an Automated Build Process.

A **version control system** manages changes to your source code over time.

- Git
- Apache Subversion
- Team Foundation Version Control

A **package management system** install uninstalls, and manages software packages.

- NuGet
- Node Package Manager (npm)
- Chocolatey
- HomeBrew
- RPM

A **continuous integration system** merges all developer working copies into a shared mainline several times daily.

- Azure DevOps
- TeamCity
- Jenkins

An **automated build process** creates a software build, including compiling, packaging, and running automated tests.

- Apache Ant
- NAnt2
- Gradle

#### Note

Your team needs to select the specific platforms and tools they'll use for each element. You need to ensure that you've established each pillar before proceeding.

## Explore benefits of continuous integration

Completed 100 XP

- 3 minutes

Continuous integration (CI) provides many benefits to the development process, including:

- Improving code quality based on rapid feedback

- Triggering automated testing for every code change
- Reducing build times for quick feedback and early detection of problems (risk reduction)
- Better managing technical debt and conducting code analysis
- Reducing long, complex, and bug-inducing merges
- Increasing confidence in codebase health long before production deployment

## Key Benefit: Rapid Feedback for Code Quality

Possibly the most essential benefit of continuous integration is rapid feedback to the developer.

If the developer commits something and breaks the code, they'll know that immediately from the build, unit tests, and other metrics.

Suppose successful integration is happening across the team.

In that case, the developer will also know if their code change breaks something that another team member did to a different part of the codebase.

This process removes long, complex, and drawn-out bug-inducing merges, allowing organizations to deliver swiftly.

Continuous integration also enables tracking metrics to assess code quality over time. For example, unit test passing rates, code that breaks frequently, code coverage trends, and code analysis.

It can provide information on what has been changed between builds for traceability benefits. Also, introduce evidence of what teams do to have a global view of build results.

For more information, you can see: [What is Continuous Integration?](#)

## CI implementation challenges

- Have you tried to implement continuous integration in your organization?
- Were you successful?
- If you did successfully, what lessons did you learn?
- If you didn't get successful, what were the challenges?

# Describe build properties

Completed 100 XP

- 4 minutes

You may have noticed that in some demos, the build number was just an integer, yet in other demos, there's a formatted value that was based upon the date.

It is one of the items that can be set in the **Build Options**.

## Build number formatting

The example shown below is from the build options that were configured by the ASP.NET Web Application build template:

The screenshot shows the 'Build properties' section of the 'Options' tab in a build pipeline configuration. The tab bar includes 'Tasks', 'Variables', 'Triggers', 'Options' (which is underlined), 'Retention', and 'History'. Below the tab bar, there is a 'Save & queue' button, a 'Discard' button, and a 'Summary' link. The 'Build properties' section has a sub-header 'Define general build pipeline settings'. It contains a 'Description' field (empty) and a 'Build number format' field containing the value '\$(date:yyyyMMdd)\$(rev:.r)'. A help icon (info symbol) is located next to the 'Build number format' label.

In this case, the date has been retrieved as a system variable, then formatted via yyyyMMdd, and the revision is then appended.

## Build status

While we have been manually queuing each build, we'll soon see that builds can be automatically triggered.

It's a key capability required for continuous integration.

But there are times that we might not want the build to run, even if it's triggered.

It can be controlled with these settings:

The new build request is processing

- Enabled - queue and start builds when eligible agent(s) available
- Paused - queue new builds but do not start
- Disabled - do not queue new builds

### Note

You can use the **Paused** setting to allow new builds to queue but to hold off then starting them.

## Authorization and timeouts

You can configure properties for the build job as shown here:

### Build job

Define build job authorization and timeout settings

Build job authorization scope (i)

Project collection

Build job timeout in minutes (i)

60

Build job cancel timeout in minutes (i)

5

The authorization scope determines whether the build job is limited to accessing resources in the current project. Or accessing resources in other projects in the project collection.

The build job timeout determines how long the job can execute before being automatically canceled.

A value of zero (or leaving the text box empty) specifies that there's no limit.

The build job cancel timeout determines how long the server will wait for a build job to respond to a cancellation request.

## Badges

Some development teams like to show the state of the build on an external monitor or website.

These settings provide a link to the image to use for it. Here's an example Azure Pipelines badge that has Succeeded:



For more information, see [Build Pipeline Options](#).

# Enable Continuous Integration with Azure Pipelines

Completed 100 XP

- 45 minutes

**Estimated time:** 45 minutes.

**Lab files:** none.

## Scenario

In this lab, you will learn how to define build pipelines in Azure DevOps using YAML. The pipelines will be used in two scenarios:

- As part of Pull Request validation process.
- As part of the Continuous Integration implementation.

# Objectives

After completing this lab, you'll be able to:

- Include build validation as part of a Pull Request.
- Configure CI pipeline as code with YAML.

# Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).

# Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Include build validation as part of a Pull Request.
- Exercise 2: Configure CI Pipeline as Code with YAML.

# Describe pipelines and concurrency

## Understand parallel jobs

Completed 100 XP

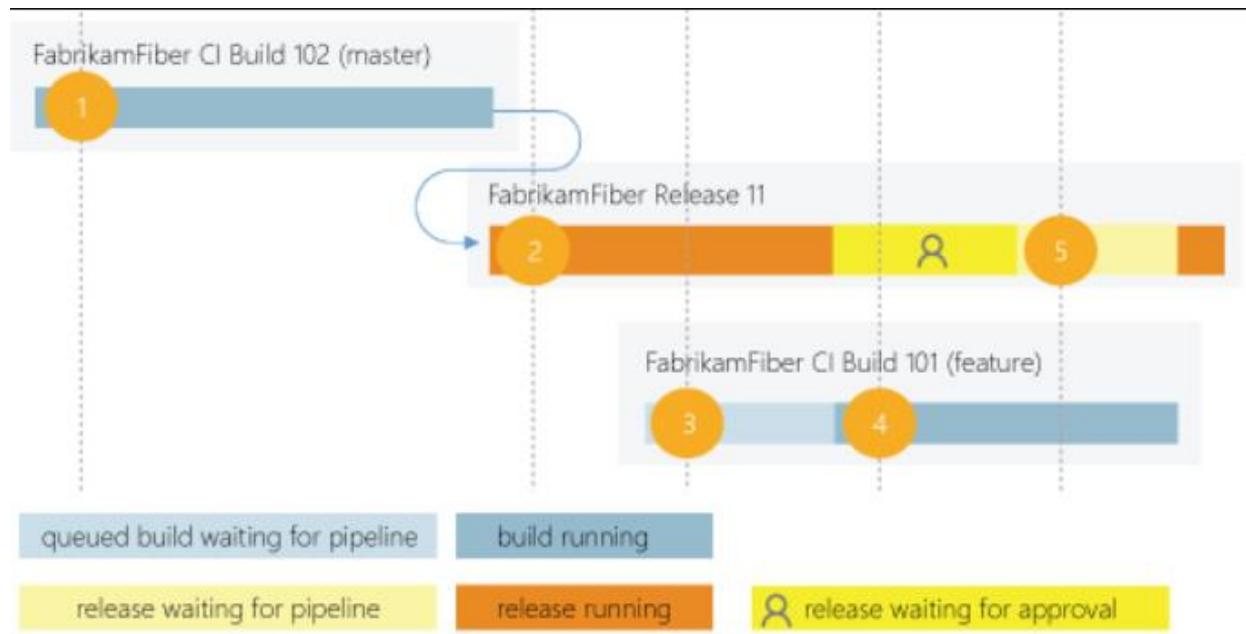
- 2 minutes

### How a parallel job is consumed by a build or release

Consider an organization that has only one Microsoft-hosted parallel job.

This job allows users in that organization to collectively run only one build or release job at a time.

When more jobs are triggered, they're queued and will wait for the previous job to finish.



A release consumes a parallel job only when it's being actively deployed to a stage.

While the release is waiting for approval or manual intervention, it doesn't consume a parallel job.

## A simple example of parallel jobs

- FabrikamFiber CI Build 102 (main branch) starts first.
- Deployment of FabrikamFiber Release 11 is triggered by the completion of FabrikamFiber CI Build 102.
- FabrikamFiber CI Build 101 (feature branch) is triggered. The build can't start yet because Release 11's deployment is active. So, the build stays queued.
- Release 11 waits for approvals. Fabrikam CI Build 101 starts because a release waiting for approvals doesn't consume a parallel job.
- Release 11 is approved. It resumes only after Fabrikam CI Build 101 is completed.

## Relationship between jobs and parallel jobs

The term job can refer to multiple concepts, and its meaning depends on the context:

- When you define a build or release, you can define it as a collection of jobs. When a build or release runs, you can run multiple jobs as part of that build or release.
- Each job consumes a parallel job that runs on an agent. When there aren't enough parallel jobs available for your organization, then the jobs are queued up and run one after the other.

You don't consume any parallel jobs when you run a server job or deploy to a deployment group.

## Estimate parallel jobs

Completed 100 XP

- 3 minutes

### Determine how many parallel jobs you need

You could begin by seeing if the free tier offered in your organization is enough for your teams.

When you've reached the 1,800 minutes per month limit for the free tier of Microsoft-hosted parallel jobs, you can start by buying one parallel job to remove this monthly time limit before deciding to purchase more.

As the number of queued builds and releases exceeds the number of parallel jobs you have, your build and release queues will grow longer.

When you find the queue delays are too long, you can purchase extra parallel jobs as needed.

## Simple estimate

A simple rule of thumb: Estimate that you'll need one parallel job for every four to five users in your organization.

## Detailed estimate

In the following scenarios, you might need multiple parallel jobs:

- If you have multiple teams, and if each of them requires a CI build, you'll likely need a parallel job for each team.
- If your CI build trigger applies to multiple branches, you'll likely need a parallel job for each active branch.
- If you develop multiple applications by using one organization or server, you'll likely need more parallel jobs: one to deploy each application simultaneously.

## View available parallel jobs

Browse to Organization settings > Pipelines > Parallel jobs.

## Location of parallel jobs in organization settings

URL example: [https://{{your\\_organization}}/\\_settings/buildqueue?\\_a=concurrentJobs](https://{{your_organization}}/_settings/buildqueue?_a=concurrentJobs)

View the maximum number of parallel jobs that are available in your organization.

Select View in-progress jobs to display all the builds and releases that are actively consuming an available parallel job or queued waiting for a parallel job to be available.

The screenshot shows the 'Organization Settings' page under 'Azure Active Directory'. The left sidebar lists various settings categories. The 'Parallel jobs' section is highlighted. The main content area is divided into 'Private projects' and 'Public projects'. Under 'Private projects', there's a summary for 'Microsoft-hosted' parallel jobs, indicating a 'Free tier' limit of 1 parallel job up to 1800 mins/mo, with 0/1800 minutes consumed. A 'Purchase parallel jobs' button is available. Below this, a table shows usage details: 3 parallel jobs, 1 free parallel job, 2 Visual Studio Enterprise subscribers, and a monthly purchases section with a 'Change' link. Under 'Public projects', there's a summary for 'Microsoft-hosted' parallel jobs, indicating a limit of 10 parallel jobs, with 0 currently used. Below this, a table shows usage details: 10 parallel jobs, and an 'Unlimited' section for parallel jobs.

| Parallel jobs                        | Limit  | Description                           |
|--------------------------------------|--|---------------------------------------|
| Microsoft-hosted                     | Free tier<br>1 parallel job up to 1800 mins/mo | Currently 0/1800 minutes are consumed |
| Self-hosted                          | 3 Parallel jobs                                |                                       |
| Free parallel jobs                   | 1  |                                       |
| Visual Studio Enterprise subscribers | 2  |                                       |
| Monthly purchases                    | 0 Change                                       |                                       |

| Parallel jobs    | Limit                   | Description                         |
|------------------|-------------------------|-------------------------------------|
| Microsoft-hosted | 10 Parallel jobs        | Currently 0/10 minutes are consumed |
| Self-hosted      | Unlimited Parallel jobs |                                     |

## Sharing of parallel jobs across projects in a collection

Parallel jobs are purchased at the organization level, and they're shared by all projects in an organization.

Currently, there isn't a way to partition or dedicate parallel job capacity to a specific project or agent pool. For example:

- You purchase two parallel jobs in your organization.
- You queue two builds in the first project, and both the parallel jobs are consumed.
- You queue a build in the second project. That build won't start until one of the builds in your first project is completed.

# Describe Azure Pipelines and open-source projects

Completed 100 XP

- 4 minutes

Azure DevOps offers developers a suite of DevOps capabilities, including Source control, Agile planning, Build, Release, Test, and more.

But to use Azure DevOps features requires the user to first sign in using a Microsoft or GitHub Account.

However, this blocks many engaging scenarios where you want to publicly share your code and artifacts or provide a wiki library or build status page for unauthenticated users.

With public projects, users can mark an Azure DevOps Team Project as public.

This will enable anonymous users to view the contents of that project in a read-only state enabling collaboration with anonymous (unauthenticated) users that wasn't possible before.

Anonymous users will essentially see the same views as authenticated users, with non-public functionality such as settings or actions (such as queue build) hidden or disabled.

## Public versus private projects

Projects in Azure DevOps provide a repository for source code and a place for a group of developers and teams to plan, track progress, and collaborate on building software solutions.

One or more projects can be defined within an organization in Azure DevOps.

Users that aren't signed into the service have read-only access to public projects on Azure DevOps.

Private projects require users to be granted access to the project and signed in to access the services.

## Supported services

Non-members of a public project will have read-only access to a limited set of services, precisely:

- Browse the code base, download code, view commits, branches, and pull requests.
- View and filter work items.
- View a project page or dashboard.
- View the project Wiki.
- Do a semantic search of the code or work items.

For more information, see [Differences and limitations for non-members of a public project](#).

## A practical example: .NET Core CLI

Supporting open-source development is one of the most compelling scenarios for public projects. A good example is the .NET Core CLI.

Their source is hosted on GitHub, and they use Azure DevOps for their CI builds.

However, if you click on the build badges in their readme, you'll not see the build results unless you were one of the project's maintainers.

Since this is an open-source project, everybody should view the full results to see why a build failed and maybe even send a pull request to help fix it.

Thanks to public projects capabilities, the team will enable just that experience. Everyone in the community will have access to the same build results, whether they are a maintainer on the project.

## How do I qualify for the free tier of Azure Pipelines for public projects?

Microsoft will automatically apply the free tier limits for public projects if you meet both conditions:

- Your pipeline is part of an Azure Pipelines public project.

- Your pipeline builds a public repository from GitHub or the same public project in your Azure DevOps organization.

## Are there limits on who can use Azure Pipelines?

You can have as many users as you want when you're using Azure Pipelines. There's no per-user charge for using Azure Pipelines.

Users with both basic and stakeholder access can author as many builds and releases as they want.

## Are there any limits on the number of builds and release pipelines that I can create?

No. You can create hundreds or even thousands of definitions for no charge. You can register any number of self-hosted agents for no cost.

## As a Visual Studio Enterprise subscriber, do I get more parallel jobs for Azure Pipelines?

Yes. Visual Studio Enterprise subscribers get one self-hosted parallel job in each Azure DevOps Services organization where they're a member.

## When you're using the per-minute plan, you can run only one job at a time.

If you run builds for more than 14 paid hours in a month, the per-minute plan might be less cost-effective than the parallel jobs model.

See [Azure DevOps Services Pricing | Microsoft Azure](#) for current pricing.

# Explore Azure Pipelines and Visual Designer

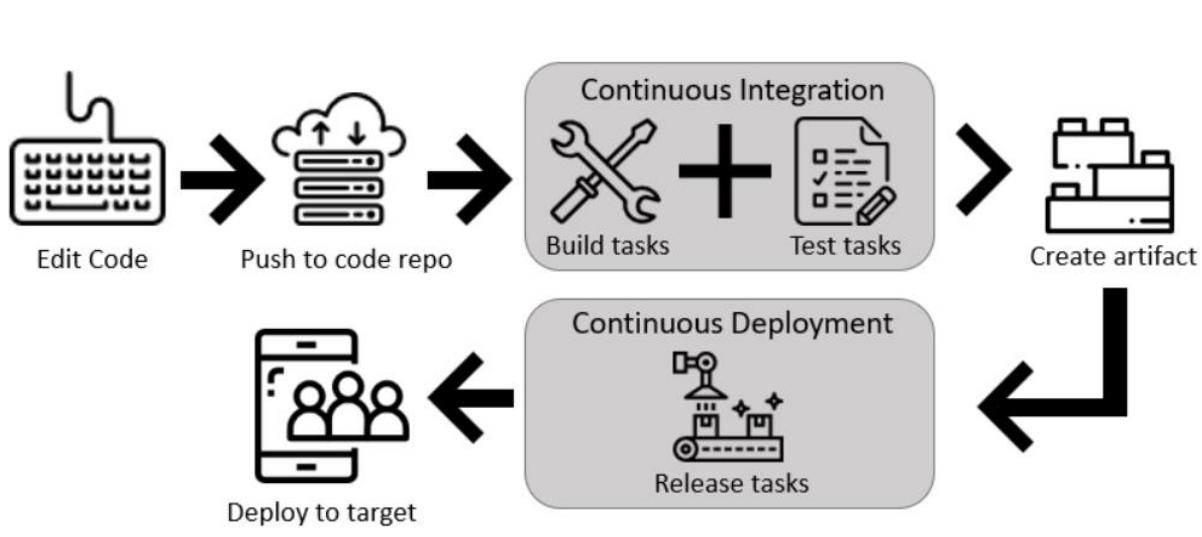
Completed 100 XP

- 2 minutes

You can create and configure your build and release pipelines in the Azure DevOps web portal with the **visual designer**. (Often referred to as "Classic Pipelines").

Configure Azure Pipelines to use your Git repo.

1. Use the Azure Pipelines visual designer to create and configure your build and release pipelines.
2. Push your code to your version control repository. This action triggers your pipeline and runs tasks such as building or testing code.
3. The build creates an artifact used by the rest of your pipeline to run tasks such as deploying to staging or production.
4. Your code is now updated, built, tested, and packaged. It can be deployed to any target.



## Benefits of using the Visual Designer

The visual designer is great for new users in continuous integration (CI) and continuous delivery (CD).

- The visual representation of the pipelines makes it easier to get started.

- The visual designer is in the same hub as the build results. This location makes it easier to switch back and forth and make changes.

If you think the designer workflow is best for you, create your first pipeline using the [visual designer](#).

## Describe Azure Pipelines and YAML

Completed 100 XP

- 3 minutes

Mirroring the rise of interest in infrastructure as code, there has been considerable interest in defining pipelines as code. However, pipeline as code doesn't mean executing a script that's stored in source control.

Codified pipelines use their programming model to simplify the setup and maximize reuse.

A typical microservice architecture will require many deployment pipelines that are identical. It's tedious to craft these pipelines via a user interface or SDK.

The ability to define the pipeline and the code helps apply all principles of code sharing, reuse, templatization, and code reviews. Azure DevOps offers you both experiences. You can either use YAML to define your pipelines or use the visual designer to do the same. You will, however, find that more product-level investments are being made to enhance the YAML pipeline experience.

When you use YAML, you define your pipeline mostly in code (a YAML file) alongside the rest of the code for your app. When using the visual designer, you define a build pipeline to build and test your code and publish artifacts.

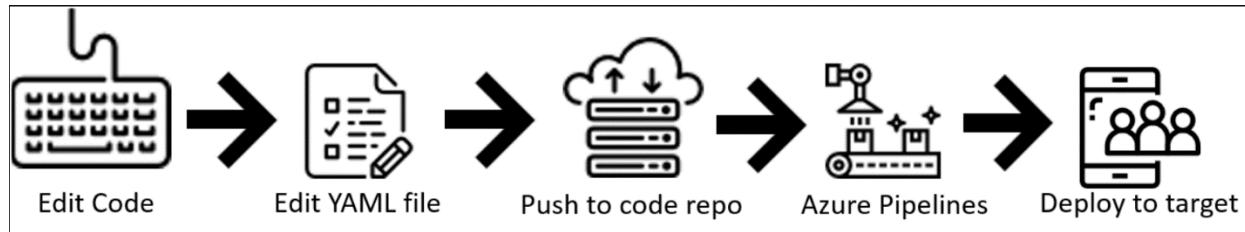
You also specify a release pipeline to consume and deploy those artifacts to deployment targets.

## Use Azure Pipelines with YAML

You can configure your pipelines in a YAML file that exists alongside your code.

1. Configure Azure Pipelines to use your Git repo.

2. Edit your azure-pipelines.yml file to define your build.
3. Push your code to your version control repository. This action kicks off the default trigger to build and deploy and then monitor the results.
4. Your code is now updated, built, tested, and packaged. It can be deployed to any target.



## Benefits of using YAML

- The pipeline is versioned with your code and follows the same branching structure. You get validation of your changes through code reviews in pull requests and branch build policies.
- Every branch you use can modify the build policy by adjusting the azure-pipelines.yml file.
- A change to the build process might cause a break or result in an unexpected outcome. Because the change is in version control with the rest of your codebase, you can more easily identify the issue.

If you think the YAML workflow is best for you, create your first pipeline by using [YAML](#).

While there's a slightly higher learning curve and a higher degree of code orientation when defining pipelines with YAML, it's now the preferred method.

# Manage Azure Pipeline agents and pools

## Choose between Microsoft-hosted versus self-hosted agents

Completed 100 XP

- 3 minutes

To build your code or deploy your software, you generally need at least one agent.

As you add more code and people, you'll eventually need more.

When your build or deployment runs, the system begins one or more jobs.

An agent is an installable software that runs one build or deployment job at a time.

### Microsoft-hosted agent

If your pipelines are in Azure Pipelines, then you've got a convenient option to build and deploy using a Microsoft-hosted agent.

With a Microsoft-hosted agent, maintenance and upgrades are automatically done.

Each time a pipeline is run, a new virtual machine (instance) is provided. The virtual machine is discarded after one use.

For many teams, this is the simplest way to build and deploy.

You can try it first and see if it works for your build or deployment. If not, you can use a self-hosted agent.

A Microsoft-hosted agent has job time limits.

### Self-hosted agent

An agent that you set up and manage on your own to run build and deployment jobs is a self-hosted agent.

You can use a self-hosted agent in Azure Pipelines. A self-hosted agent gives you more control to install dependent software needed for your builds and deployments.

You can install the agent on:

- Linux.
- macOS.
- Windows.
- Linux Docker containers.

After you've installed the agent on a machine, you can install any other software on that machine as required by your build or deployment jobs.

A self-hosted agent doesn't have job time limits.

## Explore job types

Completed 100 XP

- 1 minute

In Azure DevOps, there are four types of jobs available:

- Agent pool jobs.
- Container jobs.
- Deployment group jobs.
- Agentless jobs.

### Agent pool jobs

The most common types of jobs. The jobs run on an agent that is part of an agent pool.

### Container jobs

Similar jobs to Agent Pool Jobs run in a container on an agent part of an agent pool.

### Deployment group jobs

Jobs that run on systems in a deployment group.

## Agentless jobs

Jobs that run directly on the Azure DevOps. They don't require an agent for execution.  
It's also-often-called Server Jobs.

# Introduction to agent pools

Completed 100 XP

- 1 minute

Instead of managing each agent individually, you organize agents into agent pools. An agent pool defines the sharing boundary for all agents in that pool.

In Azure Pipelines, pools are scoped to the entire organization so that you can share the agent machines across projects.

If you create an Agent pool for a specific project, only that project can use the pool until you add the project pool into another project.

When creating a build or release pipeline, you can specify which pool it uses, organization, or project scope.

Pools scoped to a project can only use them across build and release pipelines within a project.

To share an agent pool with multiple projects, use an organization scope agent pool and add them in each of those projects, add an existing agent pool, and choose the organization agent pool. If you create a new agent pool, you can automatically grant access permission to all pipelines.

# Explore predefined agent pool

Completed 100 XP

- 1 minute

Azure Pipelines provides a pre-defined agent pool-named **Azure Pipelines** with Microsoft-hosted agents.

It will often be an easy way to run jobs without configuring build infrastructure.

The following virtual machine images are provided by default:

- Windows Server 2022 with Visual Studio 2022.
- Windows Server 2019 with Visual Studio 2019.
- Ubuntu 20.04.
- Ubuntu 18.04.
- macOS 11 Big Sur.
- macOS X Catalina 10.15.

By default, all contributors in a project are members of the User role on each hosted pool.

It allows every contributor to the author and runs build and release pipelines using a Microsoft-hosted pool.

Pools are used to run jobs. Learn about [specifying pools for jobs](#).

#### Note

See Microsoft-hosted agents for the most up-to-date list of Agent Pool Images. Also, the complete list of software installed on these machines.

## Understand typical situations for agent pools

Completed 100 XP

- 2 minutes

If you've got many agents intended for different teams or purposes, you might want to create more pools, as explained below.

## Create agent pools

Here are some typical situations when you might want to create agent pools:

- You're a project member, and you want to use a set of machines your team owns for running build and deployment jobs.
  - First, make sure you're a member of a group in All Pools with the Administrator role.
  - Next, create a New project agent pool in your project settings and select the option to Create a new organization agent pool. As a result, both an organization and project-level agent pool will be created.
  - Finally, install and configure agents to be part of that agent pool.
- You're a member of the infrastructure team and would like to set up a pool of agents for use in all projects.
  - First, make sure you're a member of a group in All Pools with the Administrator role.
  - Next, create a New organization agent pool in your admin settings and select Autoprovision corresponding project agent pools in all projects while creating the pool. This setting ensures all projects have a pool pointing to the organization agent pool. The system creates a pool for existing projects, and in the future, it will do so whenever a new project is created.
  - Finally, install and configure agents to be part of that agent pool.
- You want to share a set of agent machines with multiple projects, but not all of them.
  - First, create a project agent pool in one of the projects and select the option to Create a new organization agent pool while creating that pool.
  - Next, go to each of the other projects, and create a pool in each of them while selecting the option to Use an existing organization agent pool.
  - Finally, install and configure agents to be part of the shared agent pool.

## Communicate with Azure Pipelines

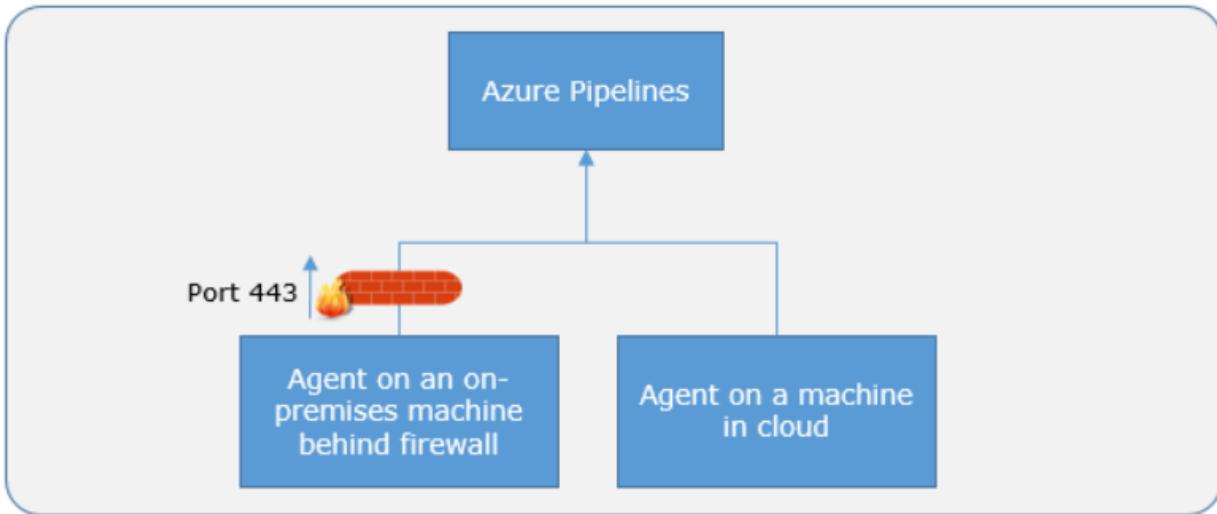
Completed 100 XP

- 2 minutes

The agent communicates with Azure Pipelines to determine which job to run and reports the logs and job status.

The agent always starts this communication. All the messages from the agent to Azure Pipelines over HTTPS depend on how you configure the agent.

This pull model allows the agent to be configured in different topologies, as shown below.



Here's a standard communication pattern between the agent and Azure Pipelines.

The user registers an agent with Azure Pipelines by adding it to an agent pool. You must be an agent pool administrator to register an agent. The identity of the agent pool administrator is needed only at the time of registration. It isn't persisted on the agent or used to communicate further between the agent and Azure Pipelines.

Once the registration is complete, the agent downloads a listener OAuth token and uses it to listen to the job queue.

Periodically, the agent checks to see if a new job request has been posted in the job queue in Azure Pipelines. The agent downloads the job and a job-specific OAuth token when a job is available. Azure Pipelines generate this token for the scoped identity specified in the pipeline. That token is short-lived and is used by the agent to access resources (for example, source code) or modify resources (for example, upload test results) on Azure Pipelines within that job.

Once the job is completed, the agent discards the job-specific OAuth token and checks if there's a new job request using the listener OAuth token.

The payload of the messages exchanged between the agent and Azure Pipelines are secured using asymmetric encryption. Each agent has a public-private key pair, and the public key is exchanged with the server during registration.

The server uses the public key to encrypt the job's payload before sending it to the agent. The agent decrypts the job content using its private key. Secrets stored in build pipelines, release pipelines, or variable groups are secured when exchanged with the agent.

## Communicate to deploy to target servers

Completed 100 XP

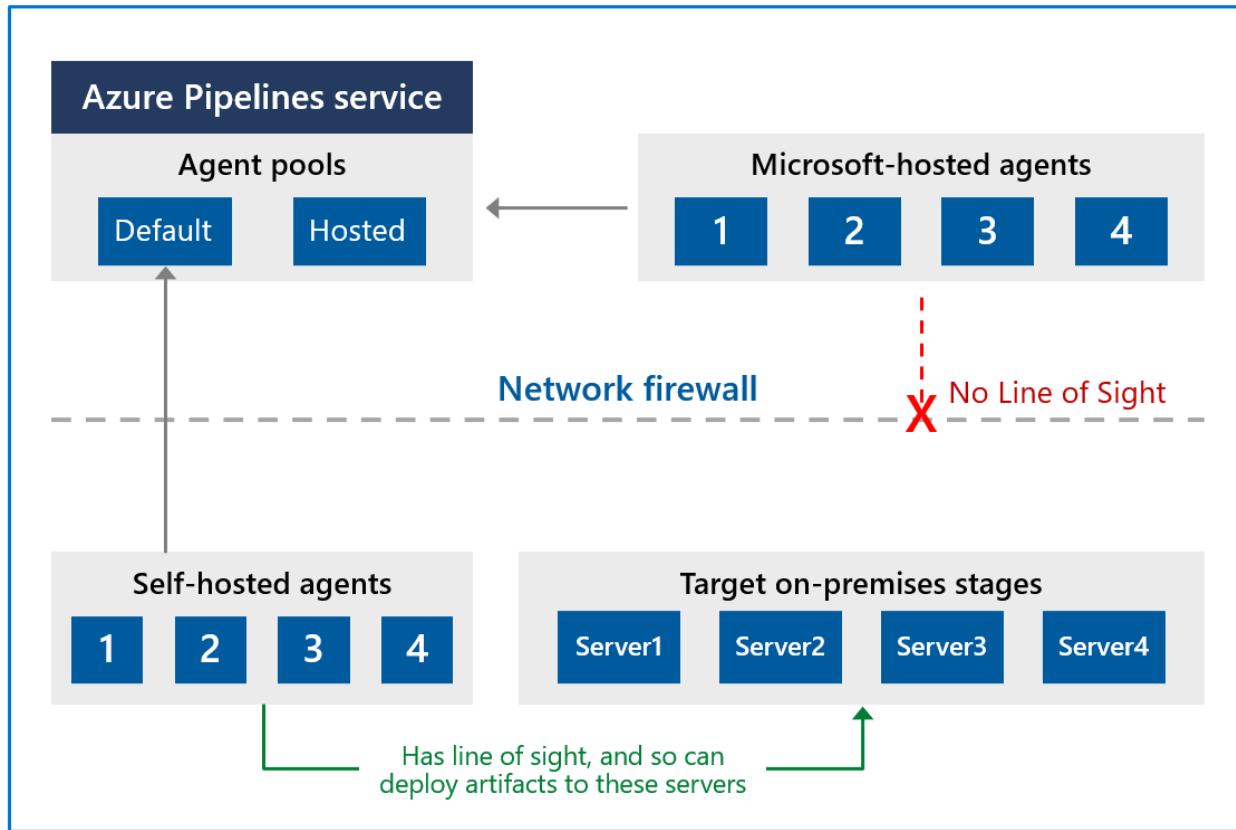
- 1 minute

When you use the agent to deploy artifacts to a set of servers, it must-have "line of sight" connectivity to those servers.

The Microsoft-hosted agent pools, by default, have connectivity to Azure websites and servers running in Azure.

Suppose your on-premises environments don't have connectivity to a Microsoft-hosted agent pool (because of intermediate firewalls). In that case, you'll need to manually configure a self-hosted agent on the on-premises computer(s).

The agents must have connectivity to the target on-premises environments and access to the Internet to connect to Azure Pipelines or Azure DevOps Server, as shown in the following diagram.



## Examine other considerations

Completed 100 XP

- 4 minutes

### Authentication

To register an agent, you need to be a member of the administrator role in the agent pool.

The identity of the agent pool administrator is only required at the time of registration. It's not persisted on the agent and isn't used in any following communication between the agent and Azure Pipelines.

Also, you must be a local administrator on the server to configure the agent.

Your agent can authenticate to Azure DevOps using one of the following methods:

## Personal access token (PAT)

Generate and use a PAT to connect an agent with Azure Pipelines. PAT is the only scheme that works with Azure Pipelines. Also, as explained above, this PAT is used only when registering the agent and not for succeeding communication.

## Interactive versus service

You can run your agent as either a service or an interactive process. Whether you run an agent as a service or interactively, you can choose which account you use to run the agent.

It's different from your credentials when registering the agent with Azure Pipelines. The choice of agent account depends solely on the needs of the tasks running in your build and deployment jobs.

For example, to run tasks that use Windows authentication to access an external service, you must run the agent using an account with access to that service.

However, if you're running UI tests such as Selenium or Coded UI tests that require a browser, the browser is launched in the context of the agent account.

After configuring the agent, we recommend you first try it in interactive mode to ensure it works. Then, we recommend running the agent in one of the following modes so that it reliably remains to run for production use. These modes also ensure that the agent starts automatically if the machine is restarted.

You can use the service manager of the operating system to manage the lifecycle of the agent. Also, the experience for auto-upgrading the agent is better when it's run as a service.

As an interactive process with autologon enabled. In some cases, you might need to run the agent interactively for production use, such as UI tests.

When the agent is configured to run in this mode, the screen saver is also disabled.

Some domain policies may prevent you from enabling autologon or disabling the screen saver.

In such cases, you may need to seek an exemption from the domain policy or run the agent on a workgroup computer where the domain policies don't apply.

### Note

There are security risks when you enable automatic login or disable the screen saver. You allow other users to walk up to the computer and use the account that automatically logs on. If you configure the agent to run in this way, you must ensure the computer is physically protected; for example, located in a secure facility. If you use Remote Desktop to access the computer on which an agent is running with autologon, simply closing the Remote Desktop causes the computer to be locked, and any UI tests that run on this agent may fail. To avoid this, use the tscon command to disconnect from Remote Desktop.

## Agent version and upgrades

Microsoft updates the agent software every few weeks in Azure Pipelines.

The agent version is indicated in the format {major}.{minor}. For instance, if the agent version is 2.1, the major version is 2, and the minor version is 1.

When a newer version of the agent is only different in minor versions, it's automatically upgraded by Azure Pipelines.

This upgrade happens when one of the tasks requires a newer version of the agent.

If you run the agent interactively or a newer major version of the agent is available, you must manually upgrade the agents. Also, you can do it from the agent pools tab under your project collection or organization.

You can view the version of an agent by navigating to Agent pools and selecting the Capabilities tab for the wanted agent.

CmdCopy

Azure Pipelines:  
[[https://dev.azure.com/{your\\_organization}/\\_admin/\\_AgentPool](https://dev.azure.com/{your_organization}/_admin/_AgentPool)] ([https://dev.azure.com/{your\\_organization}/\\_admin/\\_AgentPool](https://dev.azure.com/{your_organization}/_admin/_AgentPool))

## Question and Answer

**Do self-hosted agents have any performance advantages over Microsoft-hosted agents?**

In many cases, yes. Specifically:

- If you use a self-hosted agent, you can run incremental builds. For example, you define a CI build pipeline that doesn't clean the repo or do a clean build. Your builds will typically run faster.
  - You don't get these benefits when using a Microsoft-hosted agent. The agent is destroyed after the build or release pipeline is completed.
- A Microsoft-hosted agent can take longer to start your build. While it often takes just a few seconds for your job to be assigned to a Microsoft-hosted agent, it can sometimes take several minutes for an agent to be allocated, depending on the load on our system.

**Can I install multiple self-hosted agents on the same machine?**

Yes. This approach can work well for agents who run jobs that don't consume many shared resources. For example, you could try it for agents that run releases that mostly orchestrate deployments and don't do much work on the agent itself.

In other cases, you might find that you don't gain much efficiency by running multiple agents on the same machine.

For example, it might not be worthwhile for agents that run builds that consume many disks and I/O resources.

You might also have problems if parallel build jobs use the same singleton tool deployment, such as npm packages.

For example, one build might update a dependency while another build is in the middle of using it, which could cause unreliable results and errors.

Further instructions on how to set up self-hosted agents can be found at:

- [Self-hosted Windows agents](#).
- [Run a self-hosted agent behind a web proxy](#).

## Describe security of agent pools

Completed 100 XP

- 4 minutes

Understanding how security works for agent pools helps you control sharing and use of agents.

## Azure Pipelines

In Azure Pipelines, roles are defined on each agent pool. Membership in these roles governs what operations you can do on an agent pool.

### Note

There are differences between **Organization** and **Project** agent pools.

Expand table

| Role on an organization agent pool | Purpose   |
|------------------------------------|---|
| Reader                             | Members of this role can view the organization's agent pool and agents. You typically use it that are responsible for monitoring the agents and their health.   |
| Service Account                    | Members of this role can use the organization agent pool to create a project agent pool in accordance with the guidelines above for creating new project agent pools, you typically don't have to add a user to the organization agent pool.  |
| Administrator                      | Also, with all the above permissions, members of this role can register or unregister agents in the organization's agent pool. They can also refer to the organization agent pool when creating a project. Finally, they can also manage membership for all roles of the organization agent pool. The user that created the organization agent pool is automatically added to the Administrator role for that pool. |

The All agent pools node in the Agent Pools tab is used to control the security of all **organization** agent pools.

Role memberships for individual **organization** agent pools are automatically inherited from the 'All agent pools' node.

Roles are also defined on each organization's agent pool. Memberships in these roles govern what operations you can do on an agent pool.

Expand table

| Role on a project agent pool | Purpose   |
|------------------------------|---|
| Reader                       | Members of this role can view the project agent pool. You typically use it to add operators monitoring the build and deployment jobs in that project agent pool.  |
| User                         | Members of this role can use the project agent pool when authoring build or release pipelines.  |
| Administrator                | Also, to all the above operations, members of this role can manage membership for all roles in the project agent pool. The user that created the pool is automatically added to the Administrator role for that pool. |

The All agent pools node in the Agent pools tab controls the security of all **project** agent pools in a project.

Role memberships for individual **project** agent pools are automatically inherited from the 'All agent pools' node.

By default, the following groups are added to the Administrator role of 'All agent pools': Build Administrators, Release Administrators, Project Administrators.

# Configure agent pools and understanding pipeline styles

Completed 100 XP

- 45 minutes

**Estimated time:** 45 minutes.

**Lab files:** none.

## Scenario

YAML-based pipelines allow you to fully implement CI/CD as code, in which pipeline definitions reside in the same repository as the code that is part of your Azure DevOps project. YAML-based pipelines support a wide range of features that are part of the classic pipelines, such as pull requests, code reviews, history, branching, and templates.

Regardless of the choice of the pipeline style, to build your code or deploy your solution by using Azure Pipelines, you need an agent. An agent hosts compute resources that run one job at a time. Jobs can be run directly on the host machine of the agent or in a container. You have an option to run your jobs using Microsoft-hosted agents, which are managed for you, or implementing a self-hosted agent that you set up and manage on your own.

In this lab, you will learn how to implement and use self-hosted agents with YAML pipelines.

# Objectives

After completing this lab, you'll be able to:

- Implement YAML-based pipelines.
- Implement self-hosted agents.

# Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- [Git for Windows](#) download page. This will be installed as part of the prerequisites for this lab.
- [Visual Studio Code](#). This will be installed as part of the prerequisites for this lab.

# Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Author YAML-based Azure Pipelines.
- Exercise 2: Manage Azure DevOps agent pools.

# Introduction to continuous delivery

## Explore traditional IT development cycle

Completed 100 XP

- 2 minutes

A few years ago, IT was a facilitating department. IT was there to support the business users, and because time had proven that developed software had bad quality by default, software changes were a risk.

The resolution for this "quality problem" was to keep changes under strict control.

The department that became responsible for controlling the changes became the IT(-Pro) department.

In the past and today, the IT(-Pro) department is responsible for the stability of the systems, while the development department is responsible for creating new value.

This split brings many companies into a problematic situation. Development departments are motivated to deliver value as soon as possible to keep their customers happy.

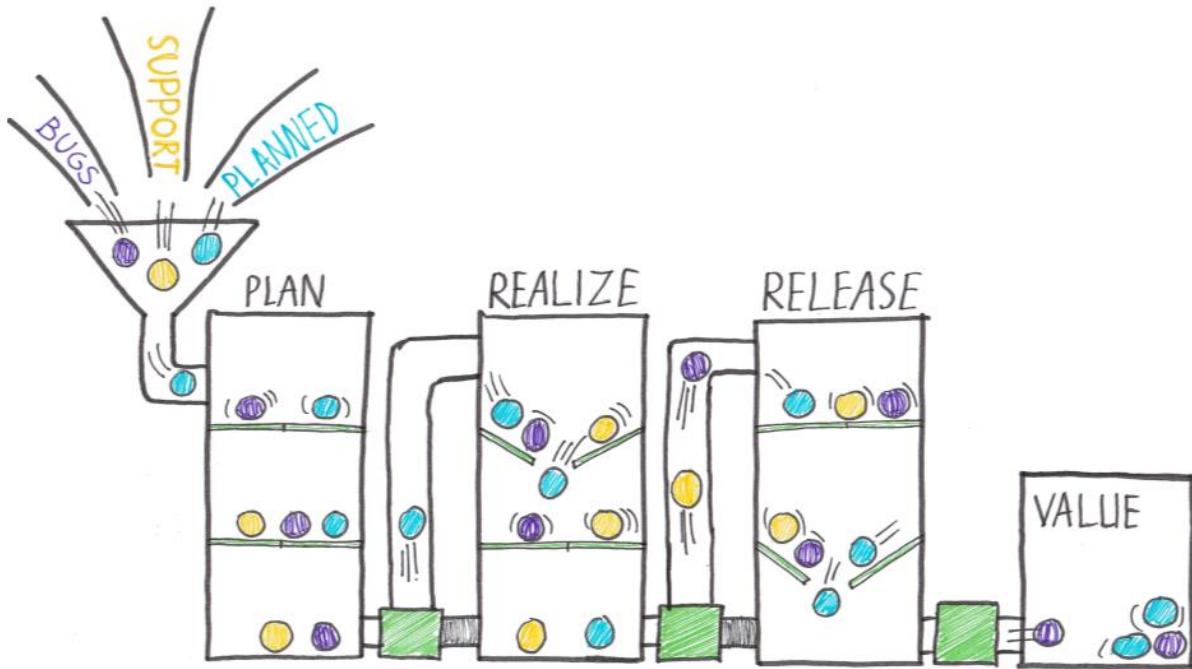
On the other hand, IT is motivated to change nothing because change is a risk, and they're responsible for eliminating the risks and keeping everything stable. And what do we get out of it? Long release cycles.

## Silo-based development

Long release cycles, numerous tests, code freezes, night and weekend work, and many people ensure that everything works.

But the more we change, the more risk it leads to, and we're back at the beginning, on many occasions resulting in yet another document or process that should be followed.

It's what I call silo-based development.



If we look at this picture of a traditional, silo-based value stream, we see Bugs and Unplanned work, necessary updates or support work, and planned (value-adding) work, all added to the teams' backlog.

Everything is planned, and the first "gate" can be opened. Everything drops to the next phase. All the work, and so all the value, moves in piles to the next stage.

It moves from the Plan phase to a Realize phase where all the work is developed, tested, and documented, and from here, it moves to the release phase.

All the value is released at the same time. As a result, the release takes a long time.

## What is continuous delivery?

Completed 100 XP

- 2 minutes

Continuous delivery (CD) is a set of processes, tools, and techniques for rapid, reliable, and continuous software development and delivery.

It means that continuous delivery goes beyond the release of software through a pipeline. The pipeline is a crucial component and the focus of this course, but continuous delivery is more.

To explain a bit more, look at the eight principles of continuous delivery:

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

If we want to fulfill these eight principles, we can see that an automated pipeline doesn't suffice.

To deploy more often, we need to reconsider our:

- Software architecture (monoliths are hard to deploy).
- Testing strategy (manual tests don't scale well).
- Organization (separated business and IT departments don't work smoothly), and so forth.

This module will focus on the release management part of continuous delivery but be aware of the other changes you might find.

Find the next bottleneck in your process, solve it, learn from it, and repeat it forever.

Continuous delivery is an enabler for DevOps. DevOps focuses on organizations and bringing people together to build and run their software products.

Continuous delivery is a practice. It's being able to deliver software on-demand. Not necessarily 1000 times a day. Deploying every code change to production is what we call continuous deployment.

To do it, we need automation and a strategy.

# Move to continuous delivery

Completed 100 XP

- 3 minutes

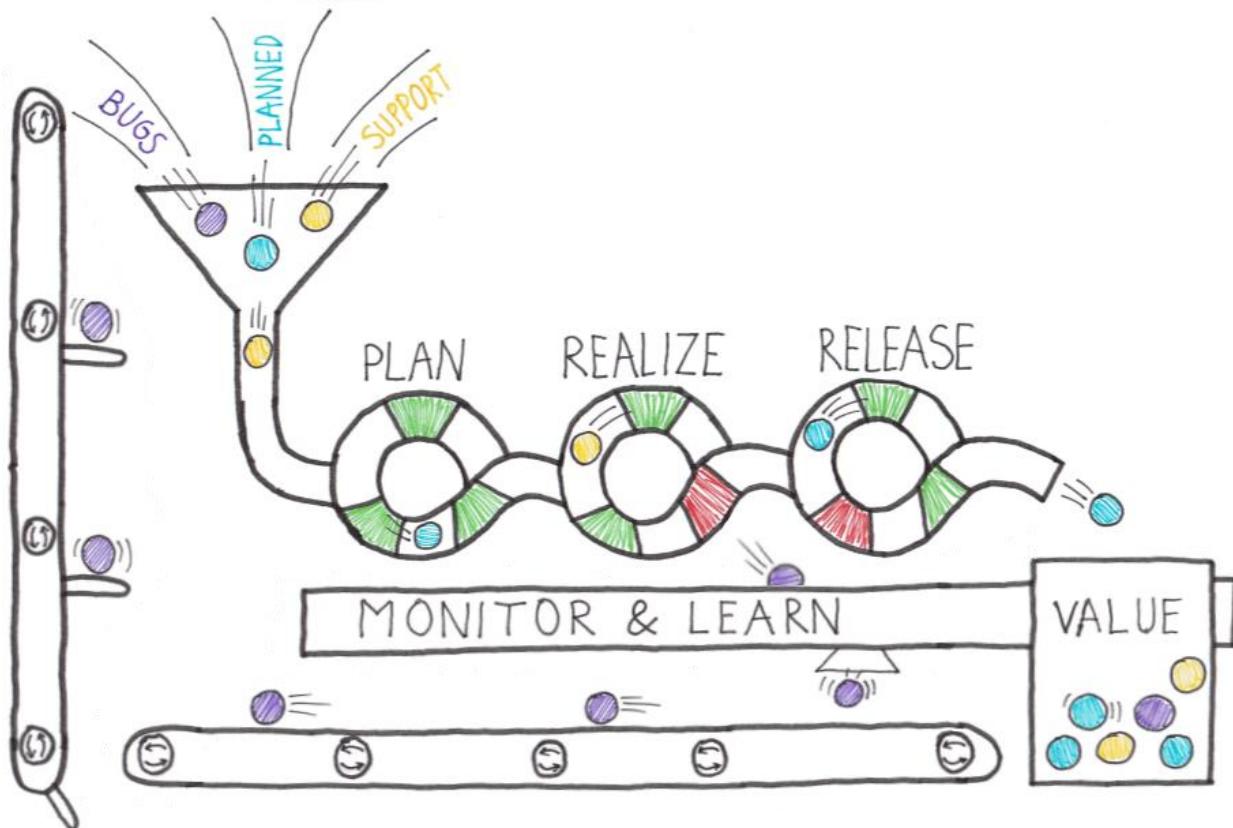
Times have changed, and we need to deal with a new normal. Our customers demand working software, and they wanted it yesterday.

If we can't deliver, they go to a competitor. And competition is fierce. With the Internet, we always have global competition.

Our competitors on our stack deliver a best-of-breed tool for one aspect of the software we built.

We need to deliver fast, and our product must be good. And we should do this with our cheap software production and high quality.

To achieve this, we need something like **Continuous Delivery**.



We need to move towards a situation where the value isn't piled up and released all at once but flows through a pipeline.

Just like in the picture, a piece of work is a marble. And only one part of the work can flow through the pipeline at once.

So, work must be prioritized in the right way. As you can see, the pipeline has green and red outlets.

We want to have these feedback loops or quality gates in place. A feedback loop can be different things:

- A unit test to validate the code.
- An automated build to validate the sources.
- An automated test on a Test environment.
- Some monitor on a server.
- Usage instrumentation in the code.

If one of the feedback loops is red, the marble can't pass the outlet and will end up in the Monitor and Learn tray.

It's where the learning happens. The problem is analyzed and solved so it's green the next time a marble passes the outlet.

Every workflow goes through the pipeline until it ends in the value tray.

The more that is automated, the faster value flows through the pipeline.

Companies want to move toward Continuous Delivery.

- They see the value.
- They hear their customers.
- Companies wish to deliver their products as fast as possible.
- Quality should be higher.
- The move to production should be faster.
- Technical Debt should be lower.

The introduction of Agile and Scrum was a great way to improve your software development practices.

Last year around 80% of companies claimed that they adopted Scrum as a software development practice.

Using Scrum, many teams can produce a working piece of software after a sprint of maybe two or three weeks.

But creating working software isn't the same as delivering working software.

The result is that all "done" increments are waiting to be delivered in the next release, which is coming in a few months.

We see now that Agile teams within a non-agile company are stuck in a delivery funnel.

The bottleneck is no longer the production of working software, but the problem has become the delivery of working software.

The finished product is waiting to be delivered to the customers to get business value, but it doesn't happen.

Continuous delivery needs to solve this problem.

## Understand releases and deployments

Completed 100 XP

- 4 minutes

One of the essential steps in moving software more quickly to production is changing how we deliver software to production.

It's common to have teams that need to do overtime on the weekend to install and release new software in our industry.

It's caused by the fact that we have two parts of the release process bolted together. As soon as we deploy new software, we also release new features to the end users.

The best way to safely move your software to production while maintaining stability is by separating these two concerns. So, we separate deployments from our release.

It can also be phrased as separating your functional release from your technical release (deployment).

**What is a release, and what is a deployment?**

When we talk about releases and deployments, we see that commonly used tools deal differently with the terminology as we did in the previous chapter.

To understand the concepts and the technical implementation in many tools, you need to know how tool vendors define the difference between a release and a deployment.

A release is a package or container containing a versioned set of artifacts specified in a release pipeline in your CI/CD process.

It includes a snapshot of 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.
- The 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).

Deployment is the action of running the tasks for one stage, which results in a tested and deployed application and other activities specified for that stage.

Starting a release starts each deployment based on the settings and policies defined in the original release pipeline.

There can be multiple deployments of each release, even for one stage.

When a release deployment fails for a stage, you can redeploy the same release to that stage.

See also [Releases in Azure Pipelines](#).

## Separating technical releases from functional releases

When we want to separate the technical and functional release, we need to start with our software itself.

The software needs to be built so that new functionality or features can be hidden from end users while it's running.

A common way to do this is the use of Feature Toggles. The simplest form of a Feature Toggle is an if statement that either executes or doesn't execute a certain piece of code.

By making the if-statement configurable, you can implement the Feature Toggle. We'll talk about Feature Toggles in Module 3 in more detail.

See also: [Explore how to progressively expose your features in production for some or all users.](#)

Once we've prepared our software, we must ensure that the installation won't expose any new or changed functionality to the end user.

When the software has been deployed, we need to watch how the system behaves. Does it act the same as it did in the past?

If the system is stable and operates the same as before, we can decide to flip a switch. It might reveal one or more features to the end user or change a set of routines part of the system.

The whole idea of separating deployment from release (exposing features with a switch) is compelling and something we want to incorporate in our Continuous Delivery practice.

It helps us with more stable releases and better ways to roll back when we run into issues when we have a new feature that produces problems.

We switch it off again and then create a hotfix. By separating deployment from the release of a feature, you make the opportunity to deploy any time of the day since the new software won't affect the system that already works.

Topics you might want to discuss are:

- 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?

# Understand release process versus release

Completed 100 XP

- 2 minutes

Before diving into high-quality release pipelines, we must consider the difference between release and release processes. Or, when you talk about tooling, a release pipeline.

We start with defining a release process or release pipeline. The release pipeline contains all the steps you walk through when you move your artifact from one of the artifact sources discussed earlier through the stages or environments.

The stage can be a development stage, a test stage, a production stage, or a stage where a specific user can access the application.

Part of your pipeline is the people who approve the release or the deployment to a specific stage. Also, triggers or schedules on which the releases execute, and the release gates, the automatic approvals of the process.

The release itself is something different. The release is an instance of the release pipeline. You can compare it with object instantiation.

In Object Orientation, a class contains the blueprint or definition of an object. But the object itself is an instance of that blueprint.



# Automate inspection of health

## Automate inspection of health

Completed 100 XP

- 2 minutes

Inspection of the release pipeline and release is something you should consider from the start.

When you run multiple deployments a day, you want to:

- Stay informed.
- Know whether a release passed or failed.
- Know the quality of the release.
- Know details about the release and how it has been done.
- Stop releases when you detect something suspicious.
- Visualize some of these things on a dashboard.

You can do a few things to stay informed about your release pipeline. In the following chapters, we'll dive a bit deeper into these.

## Release gates

Release gates allow the automatic collection of health signals from external services and then promote the release when all the signs are booming simultaneously or stop the deployment on timeout.

Typically, gates are connected with incident management, problem management, change management, monitoring, and external approval systems. Release gates are discussed in an upcoming module.

## Events, subscriptions, and notifications

Events are raised when specific actions occur, like when a release is started or a build is completed.

A notification subscription is associated with a supported event type. The subscription ensures you get notified when a specific event occurs.

Notifications are usually emails that you receive when an event occurs to which you're subscribed.

## Service hooks

Service hooks enable you to do tasks on other services when events happen in your Azure DevOps Services projects.

For example, create a card in Trello when a work item is created or send a push notification to your team's Slack when a build fails.

Service hooks can also be used in custom apps and services as a more efficient way to drive activities when events happen in your projects.

## Reporting

Reporting is the most static approach to inspection but also the most evident in many cases.

Creating a dashboard that shows the status of your build and releases combined with team-specific information is, in many cases, a valuable asset to get insights.

Read more at [About dashboards, charts, reports, & widgets](#).

## Explore events and notifications

Completed 100 XP

- 2 minutes

One of the first requests many people have when working in a system that does asynchronous actions is to get notifications or alerts. Why?

Because they don't want to open the application, log in and see if things changed repeatedly.

The ability to receive Alerts and notifications is a powerful mechanism to get notified about certain events in your system when they happen.

For example, when a build takes a while to complete, you probably don't want to stare at the screen until it has finished. But you want to know when it does.

Getting an email or another kind of notification instead is powerful and convenient. Another example is a system that needs to be monitored.

You want to get notified by the system in real time. By implementing a successful alert mechanism, you can use alerts to react to situations proactively before anybody is bothered by them.

## Alerts

However, when you define alerts, you need to be careful. When you get alerts for every single event that happens in the system, your mailbox will quickly be flooded with numerous alerts.

The more alerts you get that aren't relevant, the more significant the change that people will never look at the alerts and notifications and will miss the important ones.

## Target audience and delivery mechanism

When defining alerts or notifications, you need to think about the target audience. Who needs to react to the alerts? Don't send messages to people for information only. They'll stop looking at it quickly.

Another thing to consider when defining alerts is the mechanism to deliver them. Do you want to send an email, or do you like to send a message in Slack for your team? Or do you want to call another system to do a particular action?

Within Azure DevOps, there are multiple ways to define your alerts. By using query and filter mechanisms, you can filter out specific alerts. For example, you only want to get notified for failed releases and not for successful ones.

Almost every action in the system raises an event to which you can subscribe. A subscription is personal or for your whole team. When you have made a subscription, you can select how you want the notification to be delivered.

For more information, see also:

- [About notifications.](#)
- [Events, subscriptions, and notifications.](#)

## Explore service hooks

Completed 100 XP

- 1 minute

Service hooks enable you to do tasks on other services when your Azure DevOps Services projects happen.

For example, create a card in Trello when a work item is created.

Or send a push notification to your team's mobile devices when a build fails.

Service hooks can also be used in custom apps and services.

It's a more efficient way to drive activities when events happen in your projects.

Azure DevOps includes built-in support for the following Service Hooks:

Expand table

| Build and release. | Collaborate | Customer support | Plan and track | Integrate  |
|--------------------|-------------|------------------|----------------|------------|
| AppVeyor           | Campfire    | UserVoice        | Trello         | Azure S... |
| Bamboo             | Flowdock    | Zendesk          |                | Azure S... |
| Jenkins            | HipChat     |                  |                | Web Ho...  |
| MyGet              | Hubot       |                  |                | Zapier     |
| Slack              |             |                  |                |            |

This list will change over time.

To learn more about service hooks and how to use and create them, read [Service Hooks in Azure DevOps](#).

# Exercise - set up service hooks to monitor the pipeline

Completed 100 XP

- 4 minutes

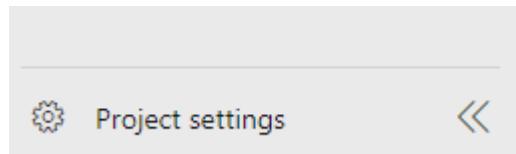
In this exercise, you'll investigate Service Hooks.

## Steps

Let's now look at how a release pipeline can communicate with other services by using service hooks.

Azure DevOps can be integrated with a wide variety of other applications. It has built-in support for many applications and generic hooks for working with other applications. Let's look.

1. Below the main menu for the **Parts Unlimited** project, click **Project Settings**.



2. In the **Project settings** menu, click **Service hooks**.

A screenshot of the 'Service Hooks' page in Azure DevOps. On the left, there is a sidebar with 'Project Settings' and a list of options: General, Overview, Teams, Security, Notifications, Service hooks (which is highlighted), and Dashboards. On the right, the main area is titled 'Service Hooks' and contains the text 'Integrate with your favorite services by notifying them when events happen in your project.' Below this is a blue button labeled '+ Create subscription' with a plus sign icon.

3. Click **+Create subscription**.

## NEW SERVICE HOOKS SUBSCRIPTION

x

# Service

Select a service to integrate with. Discover more integrations

[App Center](#)

[AppVeyor](#)

[Azuqua](#)

[Azure App Service](#)

[Azure Service Bus](#)

[Azure Storage](#)

[Bamboo](#)

[Campfire](#)

[Flowdock](#)

[Grafana](#)

[HipChat](#)

[HockeyApp](#)

[Jenkins](#)

[Microsoft Teams](#)

## App Center

App Center allows you to automate the lifecycle of your iOS, Android, Windows, and macOS apps. Connect your repo and within minutes build in the cloud, test on thousands of real devices, distribute to beta testers and app stores, and monitor real-world usage with crash and analytics data, all in one place.

### Supported events:

[Work item updated](#)

### Supported actions:

[Send notification](#)

[Learn more about this service](#)

[Previous](#)

[Next](#)

[Test](#)

[Finish](#)

[Cancel](#)

By using service hooks, we can notify other applications that an event has occurred within Azure DevOps. We could also send a message to a team in **Microsoft Teams** or **Slack**. We could also trigger an action in **Bamboo** or **Jenkins**.

4. Scroll to the bottom of the list of applications and click on **Web Hooks**.

## NEW SERVICE HOOKS SUBSCRIPTION

X

# Service

Select a service to integrate with. Discover more integrations

- Gratana
- HipChat
- HockeyApp
- Jenkins
- Microsoft Teams
- MyGet
- Office 365
- Slack
- Trello
- UserVoice
- Web Hooks
- Workplace Messaging Apps
- Zapier
- Zendesk

### Web Hooks

Provides event communication via HTTP

#### Supported events:

All events

#### Supported actions:

Post via HTTP

[Learn more about this service](#)

Previous

Next

Test

Finish

Cancel

Suppose the application that you want to communicate with isn't in the list of available application hooks.

In that case, you can almost always use the **Web Hooks** option as a generic way to communicate. It allows you to make an HTTP POST when an event occurs.

So, if, for example, you wanted to call an Azure Function or an Azure Logic App, you could use this option.

To demonstrate the basic process for calling web hooks, we'll write a message into a queue in the Azure Storage account that we have been using.

5. From the list of available applications, click **Azure Storage**.

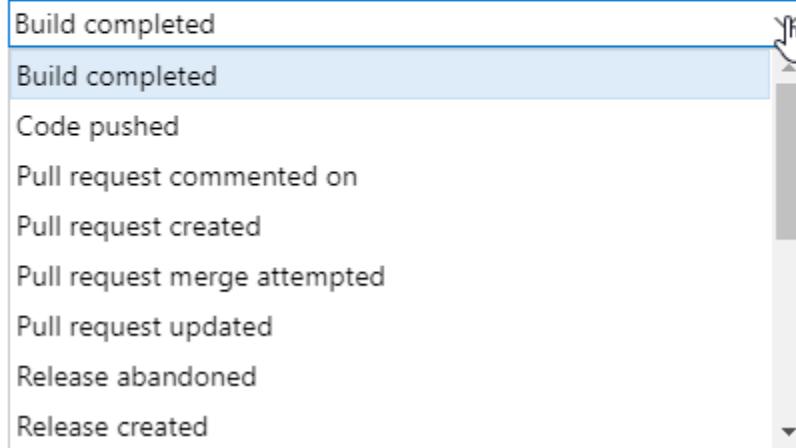
The screenshot shows a software interface titled "NEW SERVICE HOOKS SUBSCRIPTION". On the left, there is a vertical list of service names: App Center, AppVeyor, Azuqua, Azure App Service, Azure Service Bus, Azure Storage, Bamboo, Campfire, Flowdock, Grafana, HipChat, HockeyApp, Jenkins, and Microsoft Teams. The "Azure Storage" item is highlighted with a blue selection bar. To the right of this list, detailed information about "Azure Storage" is displayed. It includes a brief description: "Microsoft Azure Storage is a service for storing large numbers of messages that can be accessed from anywhere in the world. It is useful for creating a backlog of work to process asynchronously." Below this, under "Supported events:", is the option "All events". Under "Supported actions:", is the option "Insert a message in a Storage Queue". At the bottom of the dialog are buttons for "Previous", "Next", "Test", "Finish", and "Cancel".

6. Click **Next**. On the **Trigger** page, we determine which event causes the service hook to be called. Click the drop-down for **Trigger on this type of event** to see the available event types.

# Trigger

Select an event to trigger on and configure any filters.

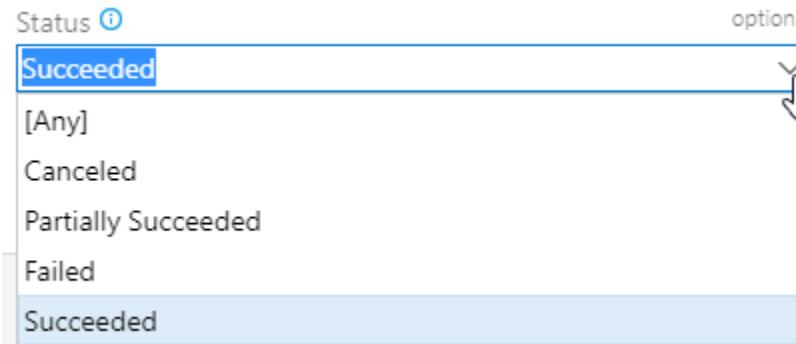
Trigger on this type of event



A screenshot of a dropdown menu titled "Trigger on this type of event". The menu lists several options: "Build completed" (selected), "Build completed", "Code pushed", "Pull request commented on", "Pull request created", "Pull request merge attempted", "Pull request updated", "Release abandoned", and "Release created". The "Build completed" option is highlighted with a blue border.

- Build completed
- Build completed
- Code pushed
- Pull request commented on
- Pull request created
- Pull request merge attempted
- Pull request updated
- Release abandoned
- Release created

7. Ensure that **Release deployment completed** is selected, then in the **Release pipeline name** select **Release to all environments**. For **Stage**, select **Production**. Drop down the list for **Status** and see the available options.



A screenshot of a dropdown menu titled "Status" with the note "optional". The menu lists several status options: "Succeeded" (selected), "[Any]", "Canceled", "Partially Succeeded", "Failed", and "Succeeded". The "Succeeded" option is highlighted with a blue border.

Status ⓘ optional

- Succeeded
- [Any]
- Canceled
- Partially Succeeded
- Failed
- Succeeded

8. Ensure that **Succeeded** is selected, then click **Next**.

## NEW SERVICE HOOKS SUBSCRIPTION

X

# Trigger

Select an event to trigger on and configure any filters.

Trigger on this type of event

Release deployment completed

**i** Remember that selected events are visible to users of the target service, even if they don't have permission to view the related artifact.

## FILTERS

Release pipeline name [i](#)

optional

Release to all environments



Stage name [i](#)

optional

Production



Status [i](#)

optional

Succeeded



Previous

Next

Test

Finish

Cancel

9. On the **Action** page, enter the name of your Azure storage account.
10. Open the Azure portal, and from the settings for the storage account, copy the value for Key in the **Access keys** section.

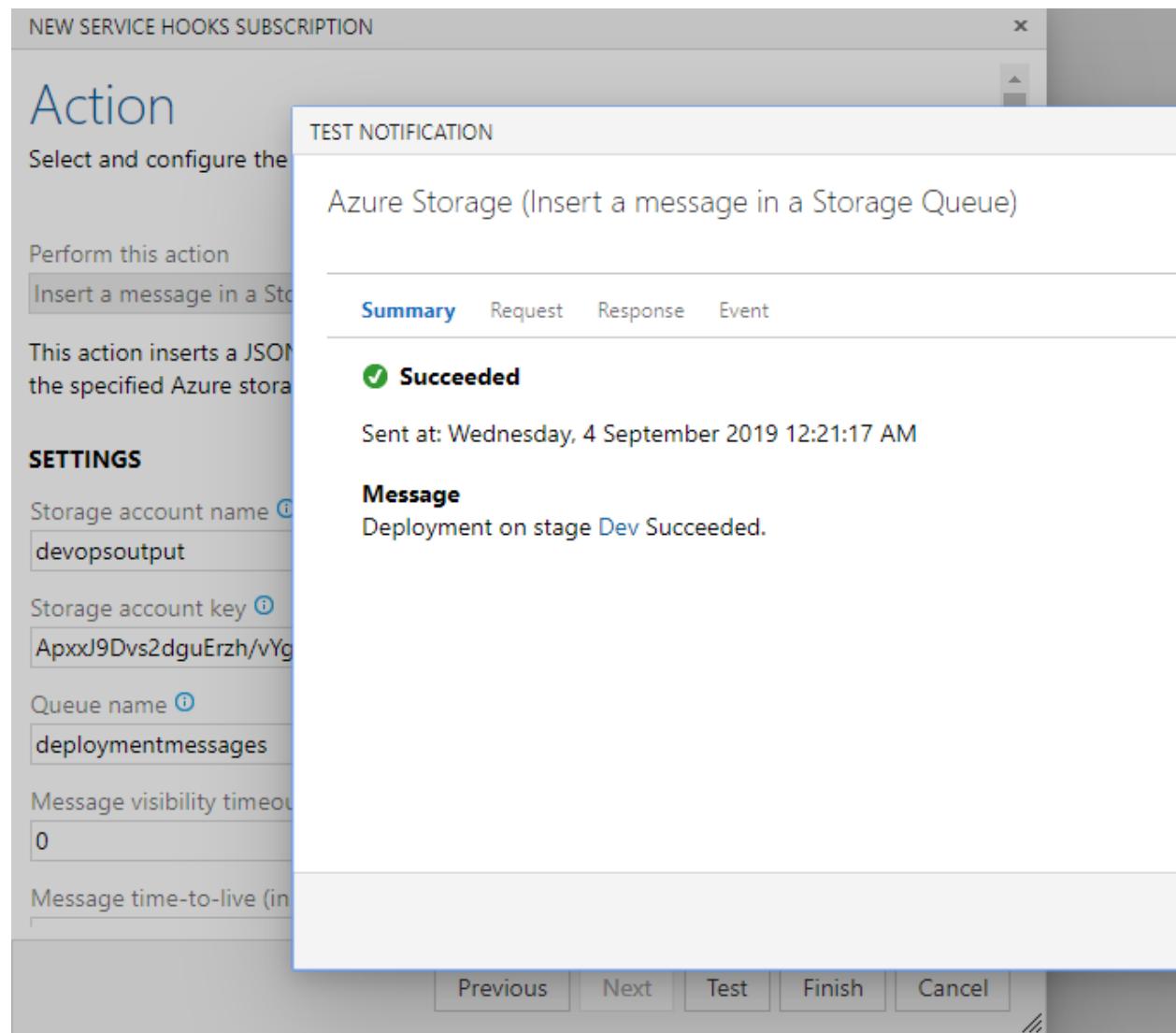
The screenshot shows the 'Access keys' section of the Azure Storage account 'devopsoutput'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Events, and Storage Explorer (preview). The 'Access keys' link is highlighted. The main content area displays a key named 'key1' with the value 'ApxxJ9Dvs2dguErzh/v'. A note states: 'Use access keys to authenticate your applications when making requests to this Azure storage account. Store your access keys securely - for example, using Azure Key Vault - and don't share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you can maintain connections using one key while regenerating the other.' It also notes: 'When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines.' Below the key, a connection string is shown: 'DefaultEndpointsProtocol=https;AccountName=devopsoutput;AccountKey=ApxxJ9Dvs2dguErzh/v'.

11. Back in the **Action** page in Azure DevOps, paste in the key.

## SETTINGS

|                                       |          |
|---------------------------------------|----------|
| Storage account name <small>i</small> | required |
| devopsoutput                          | ✓        |
| Storage account key <small>i</small>  | required |
| ApxxJ9[ 'vYgsbkz                      | ✓        |

12. For **Queue name**, enter **deploymentmessages**, then click **Test**.



13. Make sure that the test succeeded, then click **Close**, and on the **Action** page, click **Finish**.

## Service Hooks

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

+ | | | History

Consumer ↑

Event ↑

Action

Azure Storage

...

Release deployment c...

Release Release to all environm...

Insert a message in a S... Account

## Create a release to test the service hook

Now that you've successfully added the service hook, it's time to test it.

1. From the **Parts Unlimited** project's main menu, click Pipelines, click **Releases**, then click **Create release**, and in the **Create a new release** pane, enter **Test the queue service hook** for **Release description**, and click **Create**.

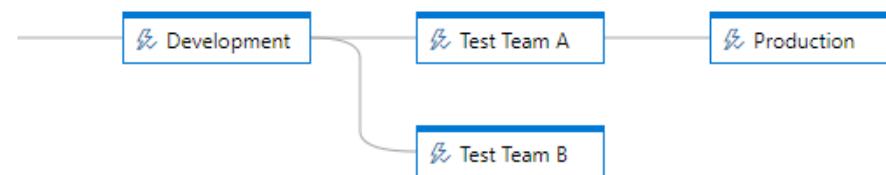
## Create a new release

X

Release to all environments

### Pipeline ^

Click on a stage to change its trigger from automated to manual.



Stages for a trigger change from automated to manual. ⓘ

### Artifacts ^

Select the version for the artifact sources for this release

| Source alias                | Version    |
|-----------------------------|------------|
| _Parts Unlimited-ASP.NET-CI | 20190901.2 |

### Release description

Test the queue service hook

Create

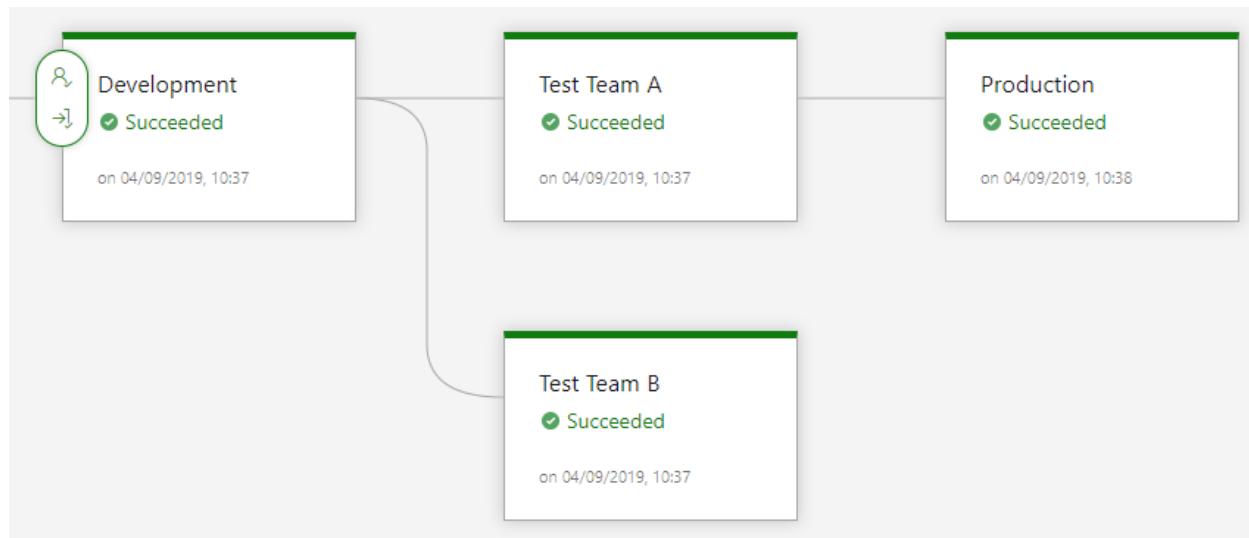
Cancel

2. Click to view the release details.



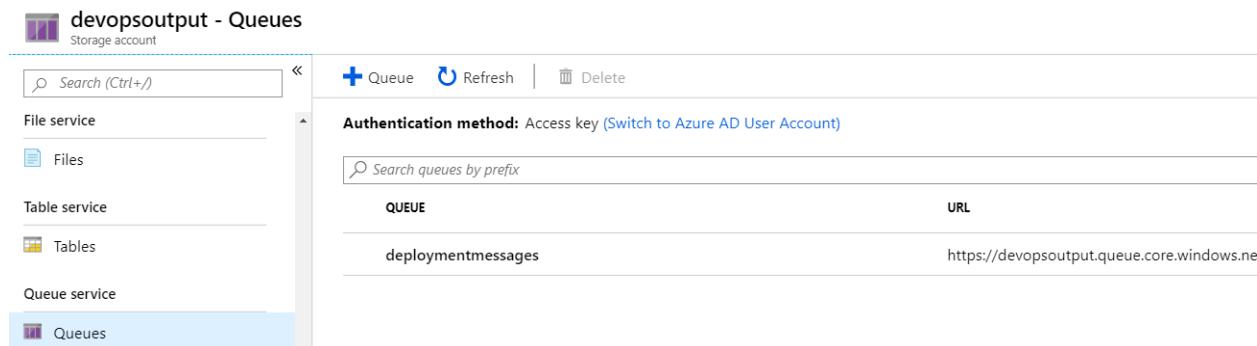
[!\[\]\(e78461a66f57eb6990ec0464a9c99f74\_img.jpg\) Search all pipelines](#)

3. If the release is waiting for approval, click to approve it and wait for the release to complete successfully.



## Check the queue contents

1. In the **Azure portal**, click **Queues** from the **Queue service** section in the blade for the storage account.



| QUEUE              | URL   |
|--------------------|---|
| deploymentmessages | <a href="https://devopsoutput.queue.core.windows.net">https://devopsoutput.queue.core.windows.net</a> |

2. Click to open the **deploymentmessages** queue.

The screenshot shows the 'deploymentmessages' queue in the Azure DevOps interface. The left sidebar has options: Overview (selected), Access Control (IAM), Settings, Access policy, and Metadata. The main area shows a table of messages with columns: ID, MESSAGE TEXT, and INSERTION TIME. There are two messages listed:

| ID                | MESSAGE TEXT  | INSERTION TIME        |
|-------------------|---|-----------------------|
| b00d5fc0-a8ed...  | {"id":"53b18aab-0f53-4bc6-9394-2bb2283c438b","eventType":"ms.vss-rel..."}   | 9/4/2019, 10:21:17 AM |
| 0ce1acee-2002-... | {"id":"48af54d0-a806-4744-9d1f-ca5f262bbcc5","eventType":"ms.vss-relea..."} | 9/4/2019, 10:38:21 AM |

## Note

If you have run multiple releases, you might have various messages.

3. Click the latest message (usually the bottom of the list) to open it and review the message properties, then close the **Message properties** pane.

## Message properties

### ID

0ce1acee-2002-4950-889d-9677518271d6

### MESSAGE BODY

```
{"id":"48af54d0-a806-4744-9d1f-ca5f262bbcc5","eventType":"ms.vss-release.deployment-completed-event","publisherId":"rm","message":{"text":"Deployment of release Release-4 on stage Production succeeded.","html":"Deployment on stage <a href='https://greglowdevopslab.visualstudio.com/Parts%20L_a=environment-summary&definitionId=2&definitionEnvironmentId=7'>Proc succeeded."},"markdown":"Deployment on stage [Production] (https://greglowdevopslab.visualstudio.com/Parts%20Unlimi\_a=environment-summary&definitionId=2&definitionEnvironmentId=7) succeeded."}, "detailedMessage": {"text": "Deployment of release Release-4 on stage Production succeeded. Time to deploy: 00:00:26."}, "html": "Deployment on stage <a
```

You've successfully integrated this message queue with your Azure DevOps release pipeline.

# Configure Azure DevOps notifications

Completed 100 XP

- 2 minutes

After defining your target audience, you need to configure your notifications. Using Azure DevOps, you can help your team stay informed about your projects' activities.

You can configure notifications to be sent based on rules or subscriptions by default, out-of-the-box (OOB), created by you or the team or group administrator.

You can get a notification when the following items change:

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

For example, you can get notified whenever your build completes, or your release fails.

There are four notification types that you can manage in Azure DevOps:

- Personal notifications.
- Team notifications.
- Project notifications.
- Global notifications.

For each notification, you have a set of specific steps to configure. The following steps show how to manage global notifications:

1. Open your Azure DevOps organization [https://dev.azure.com/\\*\\*\{organization\}\\*\\*/\\_settings/organizationOverview](https://dev.azure.com/**\{organization\}**/_settings/organizationOverview).
2. Click on Organization settings at the bottom left side.
3. Click on Global notifications under the General tab.

| Description                             | Type                               |
|---|------------------------------------|
| Build completes                         | Build completed (any project)      |
| Pull request reviewers added or removed | Pull request (any project)         |
| Pull request completion failures        | Pull request (any project)         |
| Pull request changes                    | Pull request (any project)         |
| A comment is left on a pull request     | Pull request comment (any project) |

The Default subscriptions tab lists all default global subscriptions available. The globe icon on a notification subscription indicates the subscription is a default subscription. You can view all [default notification subscriptions](#).

You can view and enable options available in the context menu (...) for each subscription.

## Note

Only Project Collection Administrators can enable/disable any default subscription in this view. Project Collection Valid Users group can only view the details of the default subscription.

In the Subscribers tab, you can see users subscribed to each notification item. The Settings section shows the *Default delivery option* setting. All teams and groups inherit this setting.

You can see how to manage your personal notifications following [manage your personal notifications](#).

For more information, see:

- [Get started with notifications in Azure DevOps - Azure DevOps](#).
- [Manage notifications for a team, project, organization, or collection - Azure DevOps](#).
- [Events, subscriptions, and notifications - Azure DevOps](#).

# Configure GitHub notifications

Completed 100 XP

- 2 minutes

Github.com notifications provide updates about the activity that you've subscribed to. You can use the notifications inbox to customize, triage, and manage your updates.

You can choose to subscribe to notifications for:

- A conversation in a specific issue, pull request, or gist.
- All activity in a repository or team discussion.
- CI activity, such as the status of workflows in repositories set up with GitHub Actions.
- Repository issues, pull requests, releases, security alerts, or discussions (if enabled).

By default, you automatically watch all repositories you create and own by your personal account and 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.

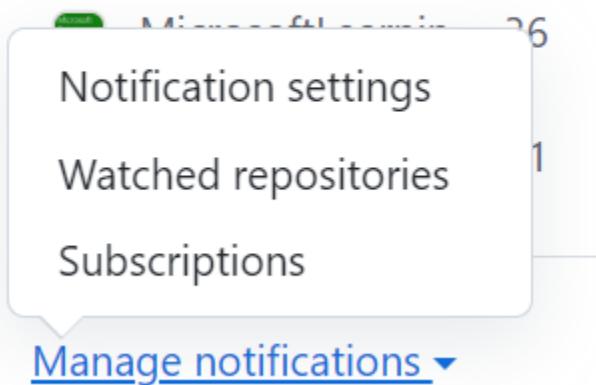
## Tip

To unsubscribe from conversations, you can change your notification settings or directly unsubscribe or unwatch activity on GitHub.com. For more information, see "[Managing your subscriptions](#)".

## Notification settings

1. Click on the notification icon in the upper-right corner of any page.
2. Click on the notification settings under the list of repositories in the left sidebar.

## Repositories



3. On the notifications settings page, choose how you receive 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.

- Email  
 Web and Mobile

### Dependabot alerts

When you're given access to [Dependabot alerts](#), automatically receive notifications when a new vulnerability is found in one of your dependencies.

- UI alerts (i)  Command Line (i)  Web

Receive security alert notifications via email

- Email each time a vulnerability is found  
 Email a digest summary of vulnerabilities

Weekly security email digest ▼

### Actions

Notifications for workflow runs on repositories set up with [GitHub Actions](#).

- Email  Web  
 Send notifications for failed workflows only

### Organization alerts

When you are given admin permissions to an organization, automatically receive notifications when a new deploy key is added.

- Email

## Email notification preferences

### Default notification email

▼

Choose which email updates you receive on conversations you're participating in or watching

- Comments on Issues and Pull Requests  
 Pull Request reviews  
 Pull Request pushes  
 Include your own updates

For more information, see:

- [About notifications - GitHub Docs](#).
- [Configuring notifications - GitHub Docs](#).
- [Viewing your subscriptions - GitHub Docs](#).

## Explore how to measure quality of your release process

Completed 100 XP

- 2 minutes

How do you measure the quality of your release process? The quality of your release process can't be measured directly because it's a process. What you can measure is how well your process works.

If your release process constantly changes, it might indicate something wrong with the process. If your releases continuously fail, and you regularly must update your release process to make it work, it might also suggest that something is wrong with your release process.

Maybe something is wrong with the schedule on which your release runs, and you notice that your release always fails on a particular day or at a specific time. Or your release always fails after the deployment to another environment. It might be an indication that some things are maybe dependent or related.

You can keep track of your release process quality by creating visualizations about the quality of all the releases following that same release process or release pipeline.

For example, we're adding a dashboard widget that shows you the status of every release.

## Release Branch Runs - Default

### Environments

|                    | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100%   | ✓<br>100% | ✓<br>100% | ► |
|--------------------|-----------|-----------|-----------|-----------|-------------|-----------|-----------|---|
| Sps.SelfTest       | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100%   | ✓<br>100% | ✓<br>100% | ► |
| Sps.SelfHost       | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100%   | ✓<br>100% | ✓<br>100% | ► |
| Tfs.SelfHost Set 1 | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100%   | ✓<br>100% | ✓<br>100% | ► |
| Tfs.SelfHost Set 2 | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✗<br>98.69% | ✓<br>100% | ✓<br>100% | ► |
| Tfs.SelfTest       | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100%   | ✓<br>100% | ✓<br>100% | ► |
| Tfs.Deploy         |           | ✓<br>100% | ✓<br>100% | ✓<br>100% |             | ✓<br>100% | ✓<br>100% |   |
| TfsOnPrem.SelfHost | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100%   | ✓<br>100% | ✓<br>100% | ► |
| TfsOnPrem.SelfTest | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100% | ✓<br>100%   | ✓<br>100% | ✓<br>100% | ► |

The release also has a quality aspect, but it's tightly related to the quality of the deployment and package deployed. When we want to measure the quality of a release itself, we can do all kinds of checks within the pipeline.

You can execute all different types of tests like integration tests, load tests, or even UI tests while running your pipeline and checking the release's quality.

Using a quality gate is also a perfect way to check the quality of your release. There are many different quality gates. For example, a gate that monitors to check if everything is healthy on your deployment targets, work item gates that verify the quality of your requirements process.

You can add extra security and compliance checks. For example, do we follow the four-eyes principle, or do we have the proper traceability?

## Examine release notes and documentation

Completed 100 XP

- 3 minutes

When you deploy a new release to a customer or install new software on your server, and you want to communicate what has been released to your customer, the usual way is to use release notes.

But where do the release notes come from? There are different ways to store your release notes.

### Document store

An often-used way of storing release notes is by creating text files or documents in some document store. This way, the release notes are stored together with other documents.

The downside of this approach is that there's no direct connection between the release in the release management tool and the release notes that belong to this release.

### Wiki

The most used way for customers is to store the release notes in a Wiki. For example:

- Confluence from Atlassian
- SharePoint Wiki
- SlimWiki
- Wiki in Azure DevOps

Release notes are created as a page in the wiki and by using hyperlinks. Relations can be associated with the build, the release, and the artifacts.

### In the codebase

When you look at it, release notes belong strictly to the release of the features you implemented and your code. In that case, the best option might be to store release notes as part of your code repository.

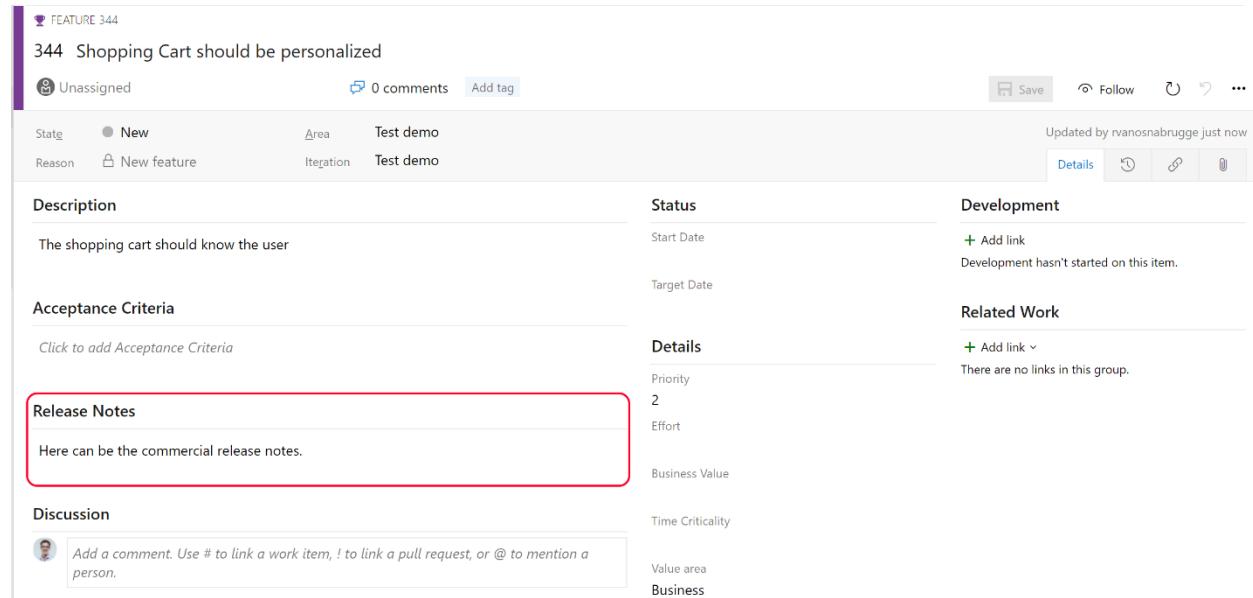
Once the team completes a feature, they or the product owner also write the release notes and save them alongside the code. This way, it becomes living documentation because the notes change with the rest of the code.

## In a work item

Another option is to store your release notes as part of your work items. Work items can be Bugs, Tasks, Product Backlog Items, or User Stories.

You can create or use a different field within the work item to save release notes in work items. In this field, you type the publicly available release notes that will be communicated to the customer.

With a script or specific task in your build and release pipeline, you can generate the release notes and store them as artifacts or publish them to an internal or external website.



The screenshot shows a Microsoft Azure DevOps work item page for a Feature named "344 Shopping Cart should be personalized". The "Release Notes" field is highlighted with a red border. The field contains the text: "Here can be the commercial release notes."

**Work Item Details:**

- State:** New
- Area:** Test demo
- Reason:** New feature
- Iteration:** Test demo

**Description:** The shopping cart should know the user

**Status:**

- Start Date
- Target Date

**Development:**

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

**Acceptance Criteria:** Click to add Acceptance Criteria

**Details:**

- Priority: 2
- Effort
- Business Value
- Time Criticality

**Related Work:**

- Add link
- There are no links in this group.

**Discussion:**

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

- [Generate Release Notes Build Task.](#)
- [Wiki Updater Tasks.](#)
- [Atlassian Confluence.](#)
- [Azure DevOps Wiki.](#)

There's a difference between functional and technical documentation. Also, a difference between documentation designing the product, primarily written upfront, and documentation describing the product afterward, like manuals or help files.

Storing technical documentation about your products in the design phase is done on a document-sharing portal, like SharePoint or Confluence.

Creating a wiki is a better and more modern way to store your documentation. Wiki's don't contain Documents, Presentations, or Spreadsheets but text files called Markdown Files.

These markdowns can refer to pictures, hold code samples, and be part of your code repository. Code repositories can deal well with text files. Changes and history can be easily tracked by using the native code tools.

However, the most significant advantage of using a Wiki instead of documents is that a Wiki is accessible to everyone in your team. People can work together on the documentation instead of waiting for each other when working on the same document by giving all the team members the correct permissions.

Manuals or documentation you release together with the product should be treated as source code. When the product changes and new functionality are added, the documentation needs to change.

You can store the documentation as part of your code repository or create a new repository containing your documentation. In any case, the documentation should be treated the same way as your source code. Create a documentation artifact in your build pipeline and deliver this artifact to the release pipeline.

The release pipeline can then deploy the documentation to a site or include it in the boxed product.

## Examine considerations for choosing release management tools

Completed 100 XP

- 5 minutes

When choosing the right Release Management tool, you should look at the possibilities of all the different components and map them to your needs.

Many tools are available in the marketplace, which we'll discuss in the next chapter. The most important thing to notice is that not every vendor or tool treats Release Management similarly.

The tools in the marketplace can be divided into two categories.

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

In many cases, companies only require the deployment part of Release Management.

Many build or release tools can perform deployment or installation. Primarily because the technical aspect of the release is executing a script or running a program, Release Management, which requires approvals, quality gates, and different stages, needs a different kind of tool that tightly integrates with the build, and CI tools aren't the same.

## Artifacts and artifact source

Artifacts can come from different sources. Treating your artifact as a versioned package needs to be stored somewhere before your release pipeline consumes it. Considerations for choosing your tool can be:

- Which Source Control systems are supported?
- Can you have one or more artifact sources in your release pipeline? In other words, can you combine artifacts from different sources into one release?
- Does it integrate with your build server?
- Does it support other build servers?
- Does it support container registries?
- How do you secure the connection to the artifact source?
- Can you extend the artifact sources?

## Triggers and schedules

Triggers are an essential component in the pipeline. Triggers are required to start a release, but if you want multiple stages, create a deployment. Considerations for choosing your trigger can be:

- Does your system support Continuous Deployment triggers?
- Can you trigger releases from an API (for integration with other tools)?
- Can you schedule releases?
- Can you schedule and trigger each stage separately?

## Approvals and gates

Starting a release using scripts, executables, or deployable artifacts doesn't differentiate between a Continuous Integration/Build tool and a Release Management tool. Adding a release approval workflow to the pipeline is the critical component that does make the difference. Considerations When it comes to approvals:

- Do you need approvals for your release pipeline?
- Is the approvers part of your company? Do they need a tool license?
- Do you want to use manual or automatic approvals? Or both?
- Can you approve an API (integration with other tools)
- Can you set up a workflow with approvers (optional and required)?
- Can you have different approvers per stage?
- Can you've more than one approver? Do you need them all to approve?
- What are the possibilities for automatic approval?
- Can you have a manual approval or step in your release pipeline?

## Stages

Running a Continuous Integration pipeline that builds and deploys your product is a commonly used scenario. But what if you want to deploy the same release to different environments? When choosing the right release management tool, you should consider the following things when it comes to stages (or environments)

- Can you use the same artifact to deploy to different stages?
- Can you differ the configuration between the stages?
- Can you have different steps for each stage?
- Can you follow the release between the stages?
- Can you track the artifacts/work items and source code between the stages?

## Build and release tasks.

Finally, the work needs to be done within the pipeline. It isn't only about the workflow and orchestration; the code must also be deployed or installed. Things to consider when it comes to the execution of tasks.

- How do you create your steps? Is it running a script (bat, shell, PowerShell CLI), or are there specialized tasks?
- Can you create your tasks?
- How do tasks authenticate to secure sources?
- Can tasks run on multiple platforms?

- Can tasks be easily reused?
- Can you integrate with multiple environments? (Linux, Windows, Container Clusters, PaaS, Cloud)
- Can you control the tasks that are used in the pipeline?

**Visual Studio | Marketplace** [Sign in](#)

Visual Studio Visual Studio Code Azure DevOps Subscriptions Build your own Publish extensions

Search Azure DevOps extensions

Showing: All categories Hosted On: Any Price: Any Sort By: Downloads

864 Results

|  |  |   |  |  |   |
|--|--|---|--|--|---|
| <b>Code Search</b><br>Microsoft <span>91.1K</span><br>Code Search provides fast, flexible and accurate search across all your code<br><br>FREE                           | <b>Test &amp; Feedback</b><br>Microsoft <span>73.4K</span><br>Now everyone on the team can own quality. Capture findings, create issues, and...<br><br>FREE        | <b>VSTS Open in Excel</b><br>Microsoft DevLabs <span>47.5K</span><br>This extension opens work items and query results in Excel from Visual Studio Team Services.<br><br>FREE | <b>Azure Artifacts</b><br>Microsoft <span>31.4K</span><br>Create, host, and share packages with your team<br><br>FREE TRIAL                        | <b>Folder Management</b><br>Microsoft DevLabs <span>29.4K</span><br>Create a folder in your source repositories from the web. No need to clone the repository...<br><br>FREE | <b>Slack Integration</b><br>Microsoft <span>26K</span><br>Keep up to date on your Azure DevOps projects from Slack<br><br>FREE                              |
| <b>Analytics</b><br>Microsoft <span>23.7K</span><br>Gain insights into the health and status of your Azure DevOps and Azure DevOps...<br><br>FREE TRIAL                  | <b>Work Item Visualization</b><br>Microsoft DevLabs <span>23.5K</span><br>Visualize relationships between work items from within the work item form.<br><br>FREE   | <b>Microsoft Teams Integration</b><br>Microsoft <span>21.1K</span><br>Microsoft Teams makes collaborating on software projects a breeze - from ide...<br><br>FREE             | <b>SonarQube</b><br>SonarSource <span>20.2K</span><br>Detect bugs, vulnerabilities and code smells across project branches and pull...<br><br>FREE | <b>Delivery Plans</b><br>Microsoft <span>19.6K</span><br>Manage your portfolio of work with a calendar based view across teams and...<br><br>FREE                            | <b>IIS Web App Deployment</b><br>Microsoft <span>18.7K</span><br>Using WinRM connect to the host Computer, to deploy a Web project using Web...<br><br>FREE |
| <b>Test Manager</b><br>Microsoft <span>17.4K</span><br>Integrated test management system for all your manual, exploratory and user...<br><br>FREE TRIAL                  | <b>Team Calendar</b><br>Microsoft DevLabs <span>15.9K</span><br>Track events important to your team, view and manage days off, quickly see when...<br><br>FREE     | <b>HockeyApp</b><br>Microsoft <span>13.7K</span><br>Distribute your builds, collect crash reports, and get feedback from your users.<br><br>FREE                              | <b>CatLight</b><br>Catlight.io <span>12.6K</span><br>Build & task status notifications in your tray<br><br>FREE                                    | <b>Replace Tokens</b><br>Guillaume Rouchon <span>12.4K</span><br>Task to replace tokens in files.<br><br>FREE  | <b>Docker Integration</b><br>Microsoft <span>11.5K</span><br>Build, push, run or deploy Docker images and multi-container Docker applications.<br><br>FREE  |
| <b>Octopus Deploy Integration</b><br>Octopus Deploy <span>10.2K</span><br>Build and Release tasks and other features for integrating with Octopus Deploy....<br><br>FREE | <b>Test Case Explorer</b><br>Microsoft DevLabs <span>10K</span><br>Manage your test cases better. Find, filter, analyze usage of test cases, and more.<br><br>FREE | <b>Release Management</b><br>Microsoft DevLabs <span>9.4K</span><br>Utility tasks for Release Management<br><br>FREE  | <b>Branch Visualization</b><br>Microsoft DevLabs <span>9.1K</span><br>Visualize your TFVC branch hierarchies<br><br>FREE                           | <b>AWS Tools for Microsoft</b><br>Amazon Web Services <span>8.7K</span><br>Tasks for Amazon S3, AWS Elastic Beanstalk, AWS CodeDeploy, AWS Lambda...<br><br>FREE             | <b>Estimate</b><br>Microsoft DevLabs <span>7.8K</span><br>Planning Poker in Visual Studio Team Services.<br><br>FREE  |

## Traceability, auditability, and security

One of the essential things in enterprises and companies that need to adhere to compliance frameworks is:

- Traceability.
- Auditability.
- Security.

Although it isn't explicitly related to a release pipeline, it's essential.

When it comes to compliance, three principles are fundamental:

- four-eyes principle
  - Does at least one other person review the deployed artifact?
  - Is the person that deploys another person the one that writes the code?
- Traceability
  - Can we see where the released software originates from (which code)?
  - Can we see the requirements that led to this change?
  - Can we follow the requirements through the code, build, and release?
- Auditability
  - Can we see who, when, and why the release process changed?
  - Can we see who, when, and why a new release has been deployed?

Security is vital in it. It isn't ok when people can do everything, including deleting evidence. Setting up the right roles, permissions, and authorization is essential to protect your system and pipeline.

When looking at an appropriate Release Management tool, you can consider the following:

- Does it integrate with your company's Active Directory?
- Can you set up roles and permissions?
- Is there a change history of the release pipeline itself?
- Can you ensure the artifact didn't change during the release?
- Can you link the requirements to the release?
- Can you link source code changes to the release pipeline?
- Can you enforce approval or the four-eyes principle?
- Can you see the release history and the people who triggered the release?

# Explore common release management tools

Completed 100 XP

- 3 minutes

The following tools are mainstream in the current ecosystem. You'll find links to the product websites to explore the product and see if it fits the needs we described in the previous chapter.

- What Artifacts and Artifact sources does the tool support?
- What Triggers and Schedules?
- Does the tool support Approvals and gates?
- Can you define multiple stages?
- How do the Build and Release Tasks work?
- How does the tool deal with Traceability, Auditability, and Security?
- What is the Extensibility model?

Per tool is indicated if it's part of a more extensive suite. Integration with a bigger suite gives you many advantages regarding traceability, security, and auditability. Numerous integrations are already there out of the box.

## GitHub Actions

GitHub Actions help you build, test, and deploy your code. You can implement continuous integration and continuous delivery (CI/CD) that allows you to make code reviews, branch management, and issue triaging work the way you want.

- Trigger workflows with various events.
- Configure environments to set rules before a job can proceed and to limit access to secrets.
- Use concurrency to control the number of deployments running at a time.

### Links

- [GitHub Actions](#).
- [Understanding GitHub Actions](#).
- [Essential features of GitHub Actions](#).

- [Deploying with GitHub Actions.](#)

## Azure Pipelines

Azure Pipelines helps you implement a build, test, and deployment pipeline for any app.

Tutorials, references, and other documentation show you how to configure and manage the continuous integration and Continuous Delivery (CI/CD) for the app and platform of your choice.

- Hosted on Azure as a SaaS in multiple regions and available as an on-premises product.
- Complete Rest API for everything around Build and Release Management.
- Integration with many build and source control systems
  - GitHub.
  - Azure Repos.
  - Jenkins.
  - Bitbucket.
  - and so on.
- Cross-Platform support, all languages, and platforms.
- Rich marketplace with extra plugins, build tasks and release tasks, and dashboard widgets.
- Part of the Azure DevOps suite. Tightly integrated.
- Fully customizable.
- Manual approvals and Release Quality Gates supported.
- Integrated with (Azure) Active Directory.
- Extensive roles and permissions.

### Links

- [Azure Pipelines.](#)
- [Building and Deploying your Code with Azure Pipelines.](#)

## Jenkins

Jenkins's leading open-source automation server provides hundreds of plugins to support building, deploying, and automating any project.

- On-premises system. They're offered as SaaS by a third party.
- No part of a bigger suite.

- Industry-standard, especially in the full-stack space.
- Integrates with almost every source control system.
- A rich ecosystem of plugins.
- CI/Build tool with deployment possibilities.
- No release management capabilities.

## Links

- [Jenkins](#).
- [Tutorial: Jenkins CI/CD to deploy an ASP.NET Core application to Azure Web App service](#).
- [Azure Friday - Jenkins CI/CD with Service Fabric](#).

## Circle CI

CircleCI's continuous integration and delivery platform help software teams rapidly release code with confidence by automating the build, test, and deploy process.

CircleCI offers a modern software development platform that lets teams ramp quickly, scale easily, and build confidently every day.

- CircleCI is a cloud-based system or an on-premises system.
- Rest API—you have access to projects, builds, and artifacts.
- The result of the build is going to be an artifact.
- Integration with GitHub and BitBucket.
- Integrates with various clouds.
- Not part of a bigger suite.
- Not fully customizable.

## Links

- [CircleCI](#).
- [How to get started on CircleCI 2.0: CircleCI 2.0 Demo](#)

## GitLab Pipelines

GitLab helps teams automate the release and delivery of their applications to shorten the delivery lifecycle, streamline manual processes and accelerate team velocity.

With Continuous Delivery (CD) built into the pipeline, deployment can be automated to multiple environments like staging and production and support advanced features such as canary deployments.

Because the configuration and definition of the application are version controlled and managed, it's easy to configure and deploy your application on demand.

Link

- [GitLab](#)

## Atlassian Bamboo

Bamboo is a continuous integration (CI) server that can automate the release management for a software application, creating a Continuous Delivery pipeline.

Link

- [Atlassian Bamboo](#)

# Manage and modularize tasks and templates

## Examine task groups

100 XP

- 1 minute

A task group allows you to encapsulate a sequence of tasks, already defined in a build or a release pipeline, into a single reusable task that can be added to a build or release pipeline, just like any other task.

You can choose to extract the parameters from the encapsulated tasks as configuration variables and abstract the rest of the task information.

Task groups are a way to standardize and centrally manage deployment steps for all your applications.

When you include a task group in your definitions and then make a change centrally to the task group, the change is automatically reflected in all the definitions that use the task group.

There's no need to change each one individually.

For more information, see [Task groups for builds and releases](#).

## Exercise - create and manage task groups

Completed 100 XP

- 4 minutes

In this exercise, you'll investigate Task Groups.

### Note

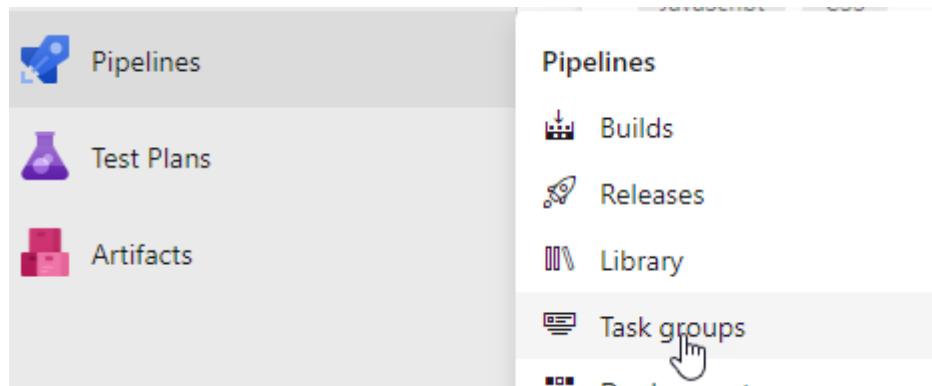
*Before starting this walkthrough, ensure you've done the steps in the prerequisites section and the previous exercises.*

## Steps

Let's now look at how a release pipeline can reuse groups of tasks.

It's common to reuse a group of tasks in more than one stage within a pipeline or in different pipelines.

1. In the main menu for the **Parts Unlimited** project, click **Pipelines**, then click **Task groups**.

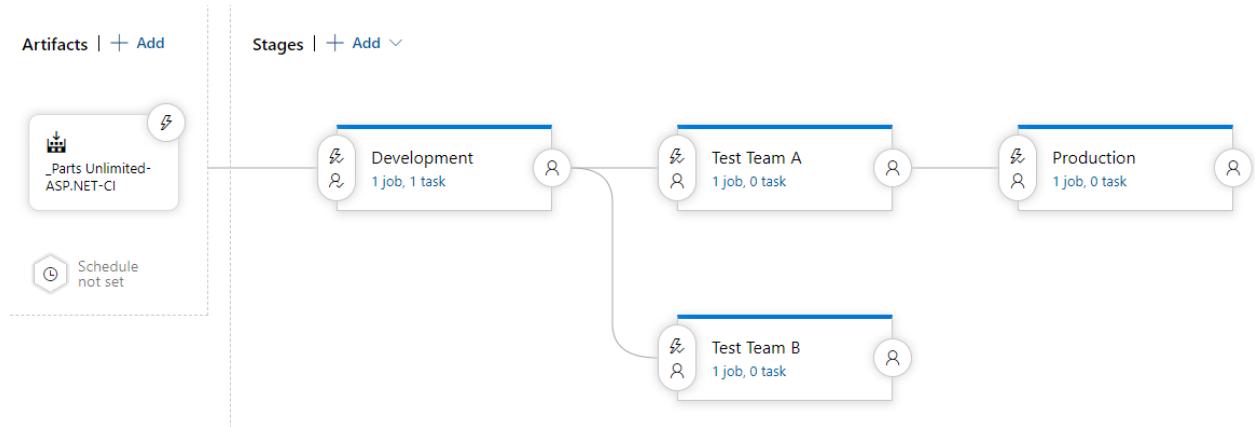


You'll notice that you don't currently have any task groups defined.

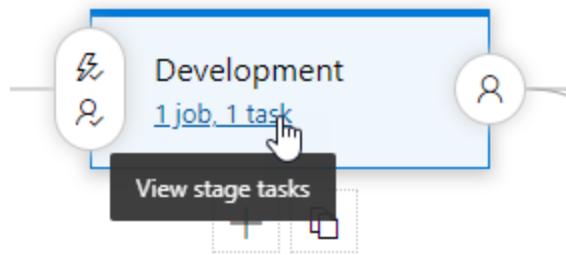
Task groups | Import Security

There's an option to import task groups, but the most common way to create a task group is directly within the release pipeline, so let's do that.

2. Click **Pipelines**, click **Releases** and click **Edit** to open the pipeline we worked on in the main menu.



- The **Development** stage currently has a single task. We'll add another task to that stage. Click the **View stage tasks** link to open the stage editor.



You can see that there's currently one task.

| Task                    | Details         |
|-------------------------|-----------------|
| Agent job               | Run on agent    |
| Backup website zip file | Azure file copy |

- Click the + sign to the right of the **Agent job** line to add a new task, in the **Search** box, type **database**.

Add tasks | Refresh

database X

-  SQL Server database deploy  
Deploy a SQL Server database using DACPAC or SQL scripts
-  Azure SQL Database deployment  
Deploy an Azure SQL Database using DACPAC or run scripts using SQLCMD
-  MySQL database deploy  
Run scripts and make changes to a MySQL Database
-  Azure Database for MySQL deployment  
Run your scripts and make changes to your Azure Database for MySQL

We'll add a task to deploy an Azure SQL Database.

5. Hover over the **Azure SQL Database Deployment** option and click **Add**. Click the **Azure SQL DacpacTask** when it appears in the list to open the settings pane.

Azure SQL Database deployment (i) View YAML Remove

Task version 1.\* ▼

Display name \*

Azure Service Connection Type  
▼

Azure Subscription \* (i) | Manage ↗  
▼ ↻

(i) This setting is required.

6. Set the **Display name** to **Deploy devopslog database**, and from the **Azure Subscriptions** drop-down list, click **ARM Service Connection**.

#### Note

*We can reuse our service connection here.*

Display name \*

Azure Service Connection Type

Azure Subscription \* ⓘ | Manage ↗

ⓘ Scoped to resource group 'SQLDEMO'

7. In the **SQL Database** section, set a unique name for the SQL Server, set the **Database to devopslog**, set the **Login** to **devopsadmin**, and set any suitable password.

SQL Database ^

Authentication Type \*

Azure SQL Server \*

Database \*

Login \*

Password \*

8. In the **Deployment Package** section, set the **Deploy type** to **Inline SQL Script**, set the **Inline SQL Script** to:

```
SQLCopy
CREATE TABLE dbo.TrackingLog
(
    TrackingLogID int IDENTITY(1,1) PRIMARY KEY,
    TrackingDetails nvarchar(max)
);
```

## Deployment Package ^

### Deploy type \*

Inline SQL Script

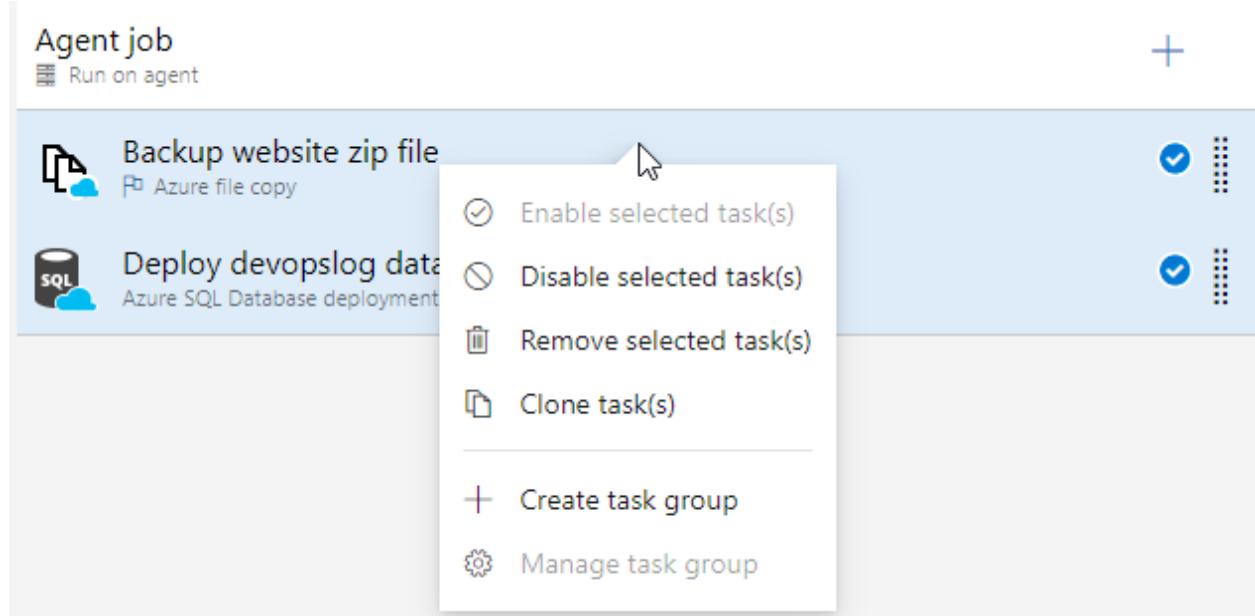
### Inline SQL Script \* ⓘ

```
CREATE TABLE dbo.TrackingLog
(
    TrackingLogID int IDENTITY(1,1) PRIMARY KEY,
    TrackingDetails nvarchar(max)
);
```

9. Click **Save** then **OK** to save the work.

Now that we have two tasks let's use them to create a task group.

10. Click to select the **Backup website zip file** task and select the **Deploy devopslog database** task, then right-click either task.



11. Click **Create task group**, then in the **Create task group** window, set **Name** to **Backup website zip file and deploy devopslog**. Click the **Category** drop-down list to see the available options. Ensure that **Deploy** is selected, and click **Create**.

Create task group

Name \*

Description

Category ⓘ

**Create** **Cancel**

The individual tasks have now disappeared from the list of tasks, and the new task group appears instead.

Development  
Deployment process ...

Agent job  
Run on agent

Task group: Backup website zip file and deploy devopslog...  
Backup website zip file and deploy devopslog database (checkmark) [More]

12. From the **Task** drop-down list, select the **Test Team A** stage.

Pipeline Tasks ▼ Variables Retention

Development Deployment process (checkmark)

Development (checkmark)

Test Team A (hand cursor)

Agent job Run on agent

Task Backup zip file log data

Production

There are currently no tasks on the stage.

Pipeline Tasks ▼ Variables Retention Options History

Test Team A Deployment process ...

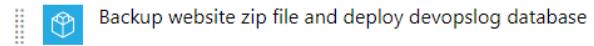
Agent job  
Run on agent

+ [More]

13. Click the + sign to the right of the **Agent job** to add a new task. In the **Search** box, type **backup** and notice that the new task group appears like any other task.

Add tasks | Refresh

backup X



14. Hover on the task group and click **Add** when it appears.

A screenshot of the Azure DevOps interface showing the 'Test Team A' stage. The 'Deployment process' tab is selected. Under 'Agent job', there's a task group titled 'Task group: Backup website zip file and deploy devopslog...'. This task group has a blue checkmark icon and three vertical dots on its right side. A '+' sign is also visible on the right side of the task group area.

Task groups allow for each reuse of a set of tasks and limit the number of places where edits need to occur.

## Walkthrough cleanup

1. Click **Remove** to remove the task group from the **Test Team A** stage.
2. From the **Tasks** drop-down list, select the **Development** stage. Again click **Remove** to remove the task group from the **Development** stage.
3. Click **Save**, then **OK**.

## Explore variables in release pipelines

Completed 100 XP

- 2 minutes

Variables give you a convenient way to get critical bits of data into various parts of the pipeline.

As the name suggests, the contents of a variable may change between releases, stages of jobs of your pipeline.

The system predefines some variables, and you're free to add your own as well.

The variable's scope is the most important thing you need to think about when using variables in the release pipeline.

You can imagine that a variable containing the target server's name may vary between a Development environment and a Test Environment.

Within the release pipeline, you can use variables in different scopes and different ways.

For more information, see [Release variables and debugging](#).

## Predefined variables

When running your release pipeline, you always need variables that come from the agent or context of the release pipeline.

For example, the agent directory where the sources are downloaded, the build number or build ID, the agent's name, or any other information.

This information is accessible in predefined variables that you can use in your tasks.

## Release pipeline variables

Choose a release pipeline variable when you need to use the same value across all the stages and tasks in the release pipeline, and you want to change the value in a single place.

## Stage variables

Share values across all the tasks within one specific stage by using stage variables.

Use a stage-level variable for values that vary from stage to stage (and are the same for all the tasks in a stage).

## Variable groups

Share values across all the definitions in a project by using variable groups. We'll cover variable groups later in this module.

## Normal and secret variables

Because the pipeline tasks are executed on an agent, variable values are passed to the various tasks using environment variables.

The task knows how to read it. You should be aware that a variable contains clear text and can be exposed to the target system.

When you use the variable in the log output, you can also see the variable's value.

When the pipeline has finished, the values will be cleared.

You can mark a variable in the release pipeline as secret. This way, the secret is hidden from the log output. It's beneficial when writing a password or other sensitive information.

The screenshot shows the 'Variables' tab in the Azure DevOps interface. At the top, there are tabs for 'Variables', 'Retention', 'Options', and 'History'. Below the tabs is a search bar labeled 'Filter by keywords' and a 'Scope' dropdown set to 'Release'. To the right of the search bar are 'List' and 'Grid' buttons. The main area displays a table of variables:

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

A red box highlights the 'Scope' column, and another red box highlights the 'Release' scope entry in the dropdown menu. The 'Scope' dropdown also includes options for 'Dev', 'Test', and 'Release'.

## Understand variable groups

Completed 100 XP

- 1 minute

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

### Examples

- 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.

For more information, see [Variable Groups for Azure Pipelines](#).

## Exercise - create and manage variable groups

Completed 100 XP

- 3 minutes

In this exercise, you'll investigate Variable Groups.

### Note

*Before starting this walkthrough, ensure you've done the steps in the prerequisites section and the previous activities.*

## Steps

Let's now look at how a release pipeline can use predefined variables, called Variable Groups.

Like how we used task groups, variable groups provide a convenient way to avoid redefining many variables when defining stages within pipelines and even when working across multiple pipelines.

Let's create a variable group and see how it can be used.

1. On the main menu for the **Parts Unlimited** project, click **Pipelines**, then click **Library**. There are currently no variable groups in the project.

## Library

Variable groups Secure files | + Variable group Security Help

2. Click **+ Variable group** to start creating a variable group. Set **Variable group name** to **Website Test Product Details**.

Library >  Website Test Product Details\*

Variable group |  Save  Clone  Security  Help

### Properties

Variable group name

Website Test Product Details

Description



Allow access to all pipelines



Link secrets from an Azure key vault as variables 

3. In the **Variables** section, click **+Add**, enter **Name**, enter **ProductCode**, and in **Value**, enter **REDPOLOXL**.

### Variables

| Name ↑  | Value     |  |
|---|-----------|---|
| ProductCode   | REDPOLOXL |   |
|  Add |           |   |

You can see an extra column that shows a lock. It allows you to have variable values that are locked and not displayed in the configuration screens.

While it's often used for values like passwords, notice an option to link secrets from an Azure key vault as variables.

It would be a preferable option for variables that provide credentials that need to be secured outside the project.

In this example, we're just providing details of a product used in testing the website.

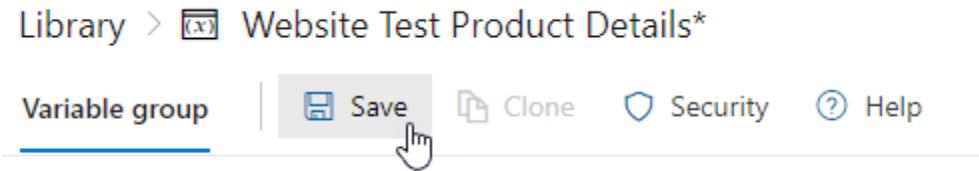
4. Add another variable called **Quantity** with a value of **12**.
5. Add another variable called **SalesUnit** with a value of **Each**.

#### Variables

| Name ↑      | Value     | ⋮ |
|-------------|-----------|---|
| ProductCode | REDPOLOXL |   |
| Quantity    | 12        |   |
| SalesUnit   | Each      |   |

[+ Add](#)

6. Click **Save** to save the new variable group.



7. On the main menu, click **Pipelines**, click Releases and click **Edit** to return to editing the release pipeline we have been working on. From the top menu, click **Variables**.

The screenshot shows the 'Variables' tab of a pipeline named 'Release to all environments'. The tab bar includes Pipeline, Tasks, Variables (underlined), Retention, Options, and History. Below the tab bar are two sections: 'Artifacts | + Add' and 'Stages | + Add ▾'. A hand cursor is hovering over the 'Variables' tab.

8. In the left-hand pane, click **Variable Groups**.

The screenshot shows the 'Variables' tab of the same pipeline. The 'Variables' tab is underlined. The left pane has three items: 'Pipeline variables', 'Variable groups' (with a hand cursor hovering over it), and 'Predefined variables'. To the right is a search bar with 'Filter by keywords' and a 'Name' input field.

Variable groups are linked to pipelines rather than being directly added to them.

9. Click **Link variable group**, then in the **Link variable group** pane, click the **Website Test Product Details** variable group (notice that it shows you how many variables are contained). In the **Variable group scope**, select the **Development**, **Test Team A**, and **Test Team B** stages.

Link variable group ⓘ

 Search

 Website Test Product Details (3)

Variable group scope

Release

Stages

 Development (+2) 

**Link**

We need the test product for development and testing, but we don't need it in production. If required in all stages, we would have chosen **Release** for the Variable group scope instead.

10. Click the **Link** to complete the link.

All pipelines >  Release to all environments

| Pipeline             | Tasks | Variables | Retention | Options | History |
|----------------------|-------|-----------|-----------|---------|---------|
| Pipeline variables   |       |           |           |         |         |
| Variable groups      |       |           |           |         |         |
| Predefined variables |       |           |           |         |         |

 Link variable group |  Manage variable groups

| Name  | Value |
|---|-------|
| Website Test Product Details (3)            |       |
| Scopes: Development,Test Team A,Test Team B |       |

The variables contained in the variable group are now available for use within all stages except production, just the same way as any other variable.

# Provision and test environments

## Provision and configure target environments

Completed 100 XP

- 5 minutes

The release pipeline deploys software to a target environment. But it isn't only the software that will be deployed with the release pipeline.

If you focus on Continuous Delivery, Infrastructure as Code and spinning up Infrastructure as part of your release pipeline is essential.

When we focus on the deployment of the Infrastructure, we should first consider the differences between the target environments that we can deploy to:

- 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.

Let us dive a bit further into these different target environments and connections.

### On-premises servers

In most cases, when you deploy to an on-premises server, the hardware and the operating system are already in place. The server is already there and ready.

In some cases, empty, but most of the time not. In this case, the release pipeline can only focus on deploying the application.

You might want to start or stop a virtual machine (Hyper-V or VMware).

The scripts you use to start or stop the on-premises servers should be part of your source control and delivered to your release pipeline as a build artifact.

Using a task in the release pipeline, you can run the script that starts or stops the servers.

To take it one step further and configure the server, you should look at technologies like PowerShell Desired State Configuration(DSC).

The product will maintain your server and keep it in a particular state. When the server changes its state, you can recover the changed configuration to the original configuration.

Integrating a tool like PowerShell DSC into the release pipeline is no different from any other task you add.

## Infrastructure as a service

When you use the cloud as your target environment, things change slightly. Some organizations lift and shift from their on-premises servers to cloud servers.

Then your deployment works the same as an on-premises server. But when you use the cloud to provide you with Infrastructure as a Service (IaaS), you can use the power of the cloud to start and create servers when needed.

It's where Infrastructure as Code (IaC) starts playing a significant role.

Creating a script or template can make a server or other infrastructural components like a SQL server, a network, or an IP address.

By defining a template or using a command line and saving it in a script file, you can use that file in your release pipeline tasks to execute it on your target cloud.

The server (or another component) will be created as part of your pipeline. After that, you can run the steps to deploy the software.

Technologies like Azure Resource Manager are great for creating Infrastructure on demand.

## Platform as a Service

When you move from Infrastructure as a Service (IaaS) to Platform as a Service (PaaS), you'll get the Infrastructure from the cloud you're running on.

For example: In Azure, you can create a Web application. The cloud arranges the server, the hardware, the network, the public IP address, the storage account, and even the web server.

The user only needs to take care of the web application on this Platform.

You only need to provide the templates instructing the cloud to create a WebApp. Functions as a Service(FaaS) or Serverless technologies are the same.

In Azure, it's called Azure Functions. You only deploy your application, and the cloud takes care of the rest. However, you must instruct the Platform (the cloud) to create a placeholder where your application can be hosted.

You can define this template in Azure Resource Manager. You can use the Azure CLI or command-line tools.

In all cases, the Infrastructure is defined in a script file and lives alongside the application code in source control.

## Clusters

Finally, you can deploy your software to a cluster. A cluster is a group of servers that host high-scale applications.

When you run an Infrastructure as a Service cluster, you must create and maintain the cluster. It means that you need to provide the templates to create a cluster.

You must also ensure you roll out updates, bug fixes, and patches to your cluster. It's comparable with Infrastructure as a Service.

When you use a hosted cluster, you should consider it a Platform as a Service. You instruct the cloud to create the cluster, and you deploy your software to the cluster.

When you run a container cluster, you can use the container cluster technologies like AKS.

## Service connections

When a pipeline needs resource access, you must often create service connections.

## Summary

Whatever the technology you choose to host your application, your Infrastructure's creation or configuration should be part of your release pipeline and source control repository.

Infrastructure as Code is a fundamental part of Continuous Delivery, allowing you to create servers and environments on demand.

## Links

- [Desired State Configuration Overview](#).
- [Azure Functions](#).
- [Azure Resource Manager](#).

## Exercise - set up service connections

Completed 100 XP

- 30 minutes

In this exercise, you'll investigate Service Connections.

### Note

To follow along with this walkthrough, you'll need an existing Azure subscription containing an existing storage account.

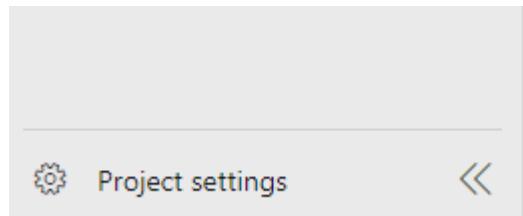
## Steps

You can set up a service connection to environments to create a secure and safe connection to the environment you want to deploy.

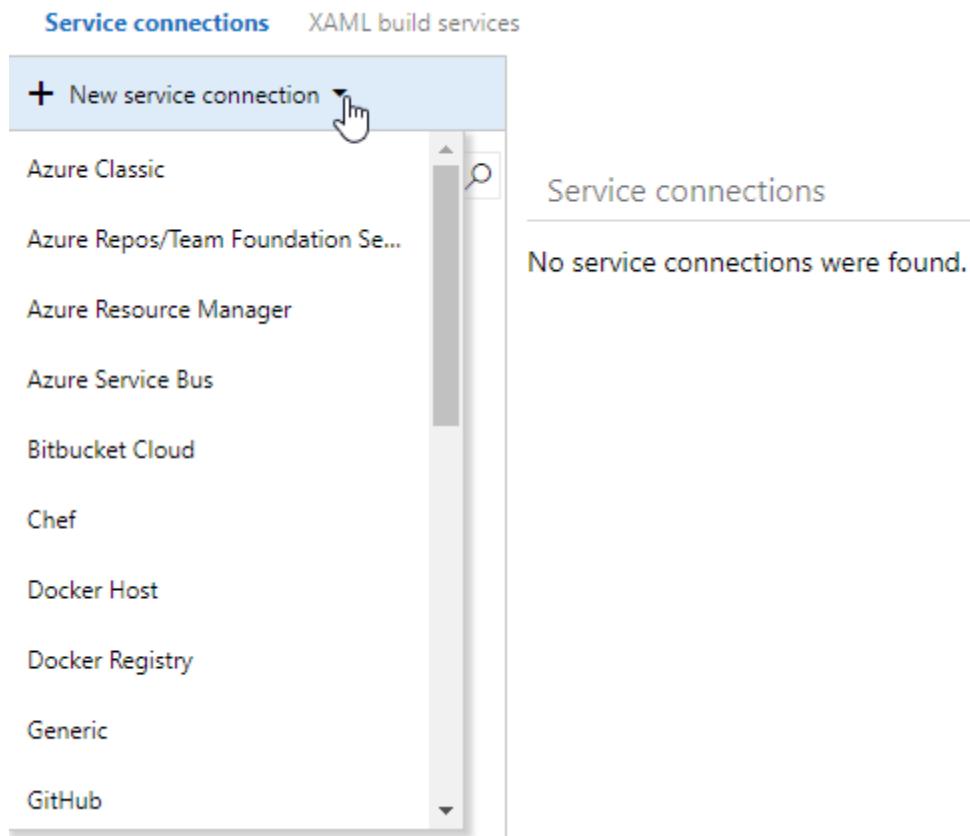
Service connections are also used to get resources from other places in a secure manner.

For example, you might need to get your source code from GitHub. Let's look at configuring a service connection to Azure in this case.

1. From the main menu in the **Parts Unlimited** project, click **Project settings** at the bottom of the screen.



2. In the Project Settings pane, from the **Pipelines** section, click **Service Connections**. Click the drop-down beside **+New service connection**.



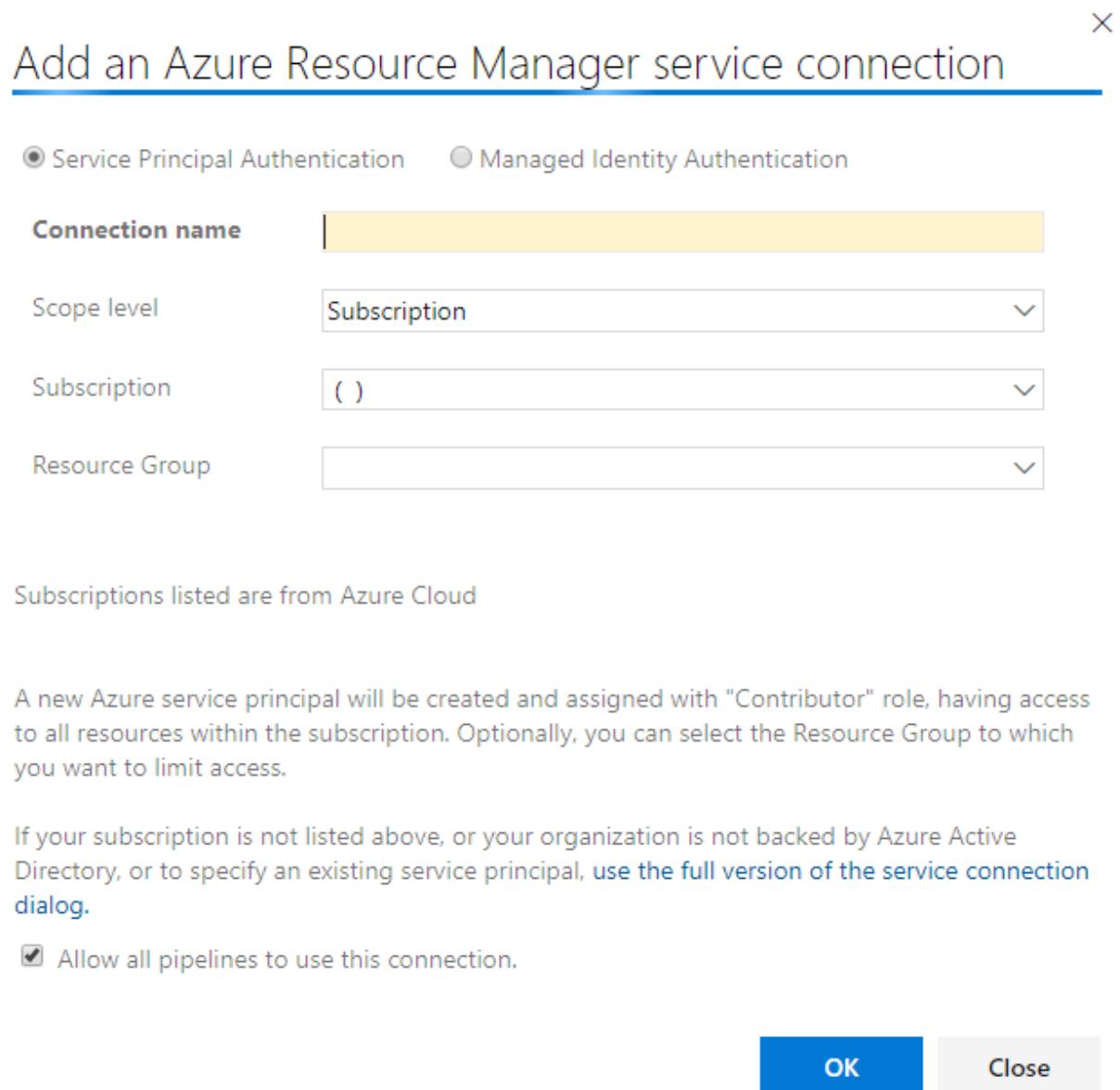
As you can see, there are many types of service connections. You can create a connection to:

- Apple App Store.
- Docker Registry

- Bitbucket.
- Azure Service bus.

In this case, we want to deploy a new Azure resource, so we'll use the Azure Resource Manager option.

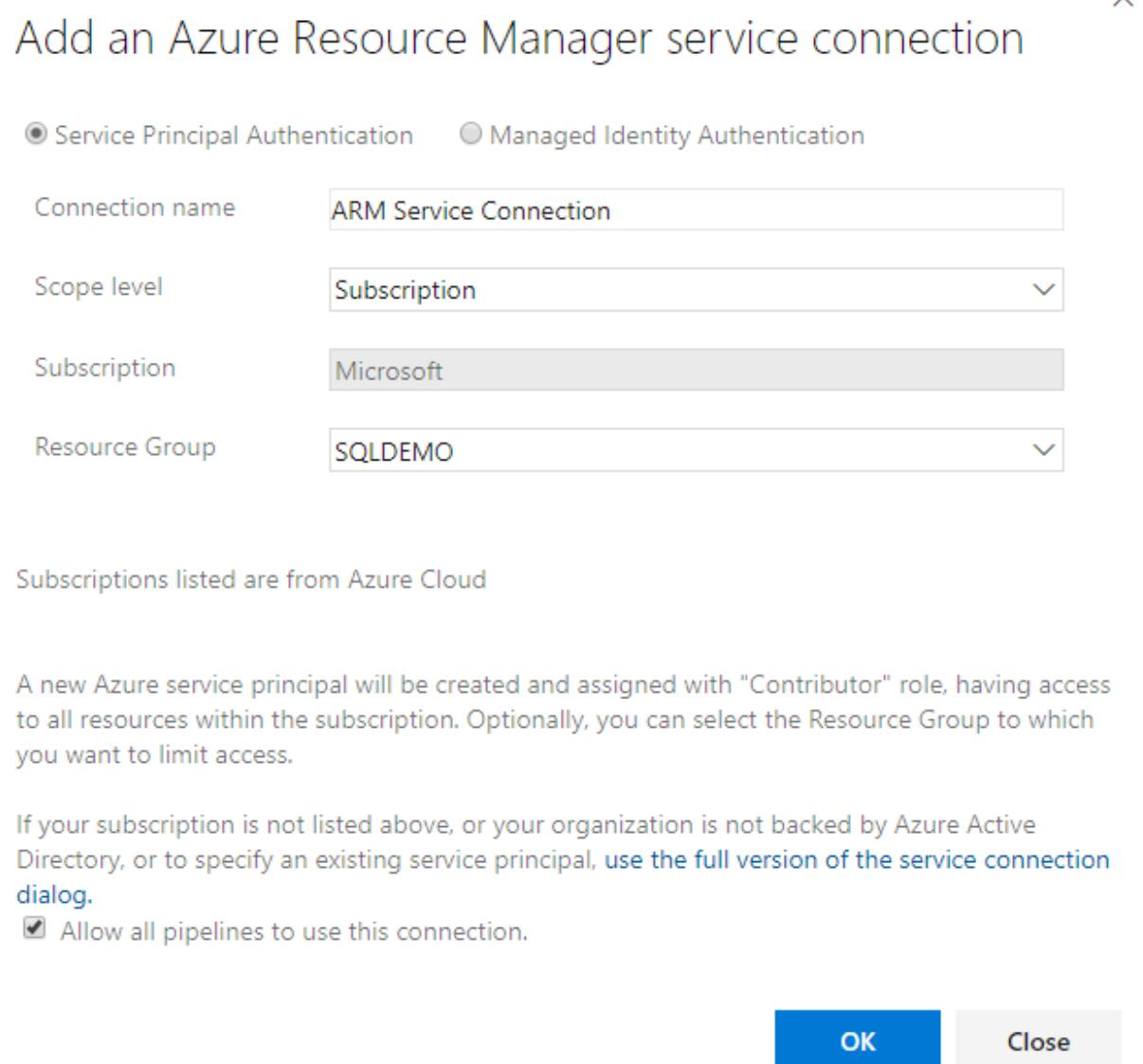
3. Click **Azure Resource Manager** to add a new service connection.



4. Set the **Connection name** to **Azure Resource Manager Service Connection**, click on an Azure **Subscription**, then select an existing **Resource Group**.

## Note

You might be prompted to sign-in Azure at this point. If so, sign-in first.



Notice that what we are creating is a **Service Principal**. We'll be using the Service Principal for authenticating to Azure. At the top of the window, there's an option to set up Managed Identity Authentication instead.

The Service Principal is a service account with only permissions in the specific subscription and resource group. It makes it a safe way to connect from the pipeline.

## Important

When you create a service connection with Azure, the service principal gets a contributor role to the subscription or resource group. It's not enough to upload data to blob storage for the service principal. You must explicitly add the service principal to the Storage Blob Data Contributor role. Otherwise, the release gets failed with an authorization permission mismatch error.

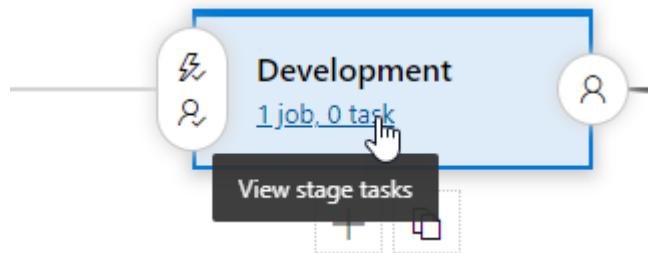
5. Click **OK** to create it. It will then be shown in the list.

The screenshot shows the 'Service connections' page in the Azure DevOps interface. On the left, there is a list of service connections, with 'ARM Service Connection' highlighted. On the right, detailed information about this connection is displayed:

- Service connection: ARM Service Connection**
- Details** tab selected, showing:
  - Type: Azure Resource Manager
  - Created by: [redacted]
  - Connected to service using Service Principal
- INFORMATION** section: [redacted]
- ACTIONS** section:
  - List of actions that can be performed on this service connection:
  - [Update service connection](#)
  - [Manage service connection roles](#)
  - [Manage Service Principal](#)
  - [Disconnect](#)

6. Click **Pipelines, Releases**, and **Edit** in the main Parts Unlimited menu to see the release pipeline. Click the link to **View stage tasks**.

## Stages | + Add ▾



The current list of tasks is then shown. Because we started with an empty template, there are no tasks yet. Each stage can execute many tasks.

## All pipelines > Release to all environments

A screenshot of the 'Release to all environments' pipeline page. At the top, there are tabs: Pipeline, Tasks (which is selected), Variables, Retention, Options, and History. Below the tabs, the 'Development' stage is listed under 'Deployment process'. The stage contains one task named 'Agent job' with the sub-label 'Run on agent'. To the right of the task is a blue '+' sign. The entire pipeline page has a light gray background.

7. Click the + sign to the right of the **Agent job** to add a new task. See the available list of task types.

A screenshot of the 'Add tasks' dialog. At the top, there are buttons for 'Add tasks' and 'Refresh', and a search bar with a magnifying glass icon. Below the search bar is a navigation bar with tabs: All, Build, Utility, Test, Package, Deploy, Tool, and Marketplace. The 'All' tab is selected. The main area lists several task types with their icons and descriptions:

- .NET Core: Build, test, package, or publish a dotnet application, or run a custom dotnet command
- Android signing: Sign and align Android APK files
- Ant: Build with Apache Ant
- App Center distribute: Distribute app builds to testers and users via Visual Studio App Center

8. In the **Search** box, enter the word **storage** and see the list of storage-related tasks. These include standard tasks and tasks available from the Marketplace.

The screenshot shows the Azure DevOps Marketplace search results for the term "storage". At the top, there is a search bar with the text "storage" and a refresh button. Below the search bar, there are two sections: "Add tasks" and "Marketplace".  
**Add tasks:** Contains a single item: "Azure file copy" (Copy files to Azure Blob Storage or virtual machines).  
**Marketplace:** Contains several items:

- Azure Storage**: Tasks to assist with the creation of storage accounts, containers therein and uploading of files.
- Azure Storage Container**: Task for creating azure storage container with a defined public access level inside an existing storage account.
- Manage Storage Account Release Tools**: Tools for managing creation Storage Accounts and its objects.
- Maven Cache**: Tasks for upload and download Maven cache from an Azure storage account.

We'll use the Azure file copy task to copy one of our source files to a storage account container.

9. Hover over the **Azure file copy** task type and click **Add** when it appears. The task will be added to the stage but requires further configuration.

The screenshot shows the Azure DevOps pipeline editor. The pipeline is named "Development" and is part of a "Deployment process". It consists of a single stage named "Agent job".  
The "Agent job" stage contains a task named "File Copy". A red warning message "Some settings need attention" is displayed next to the task icon.  
A horizontal bar at the bottom of the stage indicates that there are more tasks in the stage.

10. Click the **File Copy** task to see the required settings. Select the latest task version.

## Azure file copy ⓘ

 View YAML  Remove

 Task version 2.\* 

Display name \*

File Copy

Source \* ⓘ

This setting is required.

Azure Connection Type

Azure Resource Manager

Azure Subscription \* ⓘ | Manage ↗

ⓘ This setting is required.

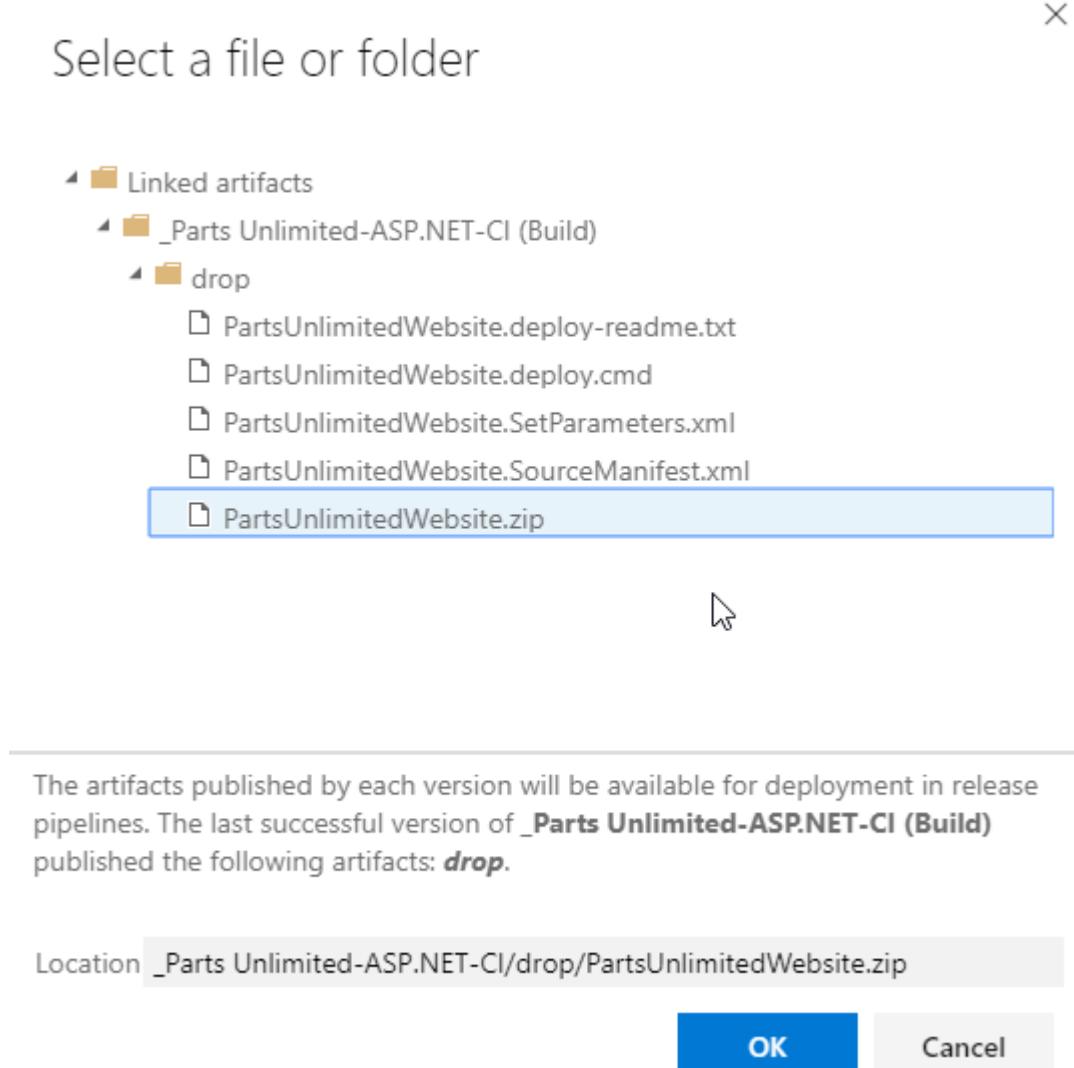
Destination Type \* ⓘ

ⓘ This setting is required.

RM Storage Account \* ⓘ

ⓘ This setting is required.

11. Set the **Display Name** to **Backup website zip file**, click the ellipsis beside **Source**, locate the file as follows, and click **OK** to select it.



We then need to provide details on connecting to the Azure subscription. The easiest and most secure way is to use our new Service Connection.

12. Find and select the **Azure Resource Manager Service Connection** we created from the **Azure Subscription** drop-down list.

Azure Subscription \* ⓘ | Manage ↗

**Available Azure service connections**

ARM Service Connection

13. From the **Destination Type** drop-down list, select **Azure Blob**, and from the **RM Storage Account** and **Container Name**, select the storage account, enter the container's name, then click **Save** at the top of the screen, **OK**.

Azure Subscription \* | Manage ▾

ARM Service Connection

Scoped to resource group 'SQLDEMO'

Destination Type \*

Azure Blob

RM Storage Account \*

devopsoutput

Container Name \*

websitezipfileoutput

14. To test the task, click **Create release**, and in the **Create a new release** pane, click **Create**.  
 15. Click the new release to view the details.

All pipelines > Release to all environments

Release [Release-3](#) has been created

Pipeline Tasks Variables Retention Options History

Development Deployment process

16. On the release page, approve the release so that it can continue.  
 17. Once the **Development** stage has been completed, you should see the file in the Azure storage account.

**Authentication method:** Access key ([Switch to Azure AD User Account](#))  
**Location:** websitezipfileoutput

Search blobs by prefix (case-sensitive)

| NAME                      | MODIFIED              | ACCESS TIER    | BLOB TYPE  | SIZE     |
|---------------------------|-----------------------|----------------|------------|----------|
| PartsUnlimitedWebsite.zip | 9/2/2019, 12:04:39 PM | Hot (Inferred) | Block blob | 9.55 MiB |

A key advantage of using service connections is that this type of connection is managed in a single place within the project settings. It doesn't involve connection details spread throughout the pipeline tasks.

## Configure automated integration and functional test automation

Completed 100 XP

- 6 minutes

The first thing that comes to mind about Continuous Delivery is that everything needs to be automated.

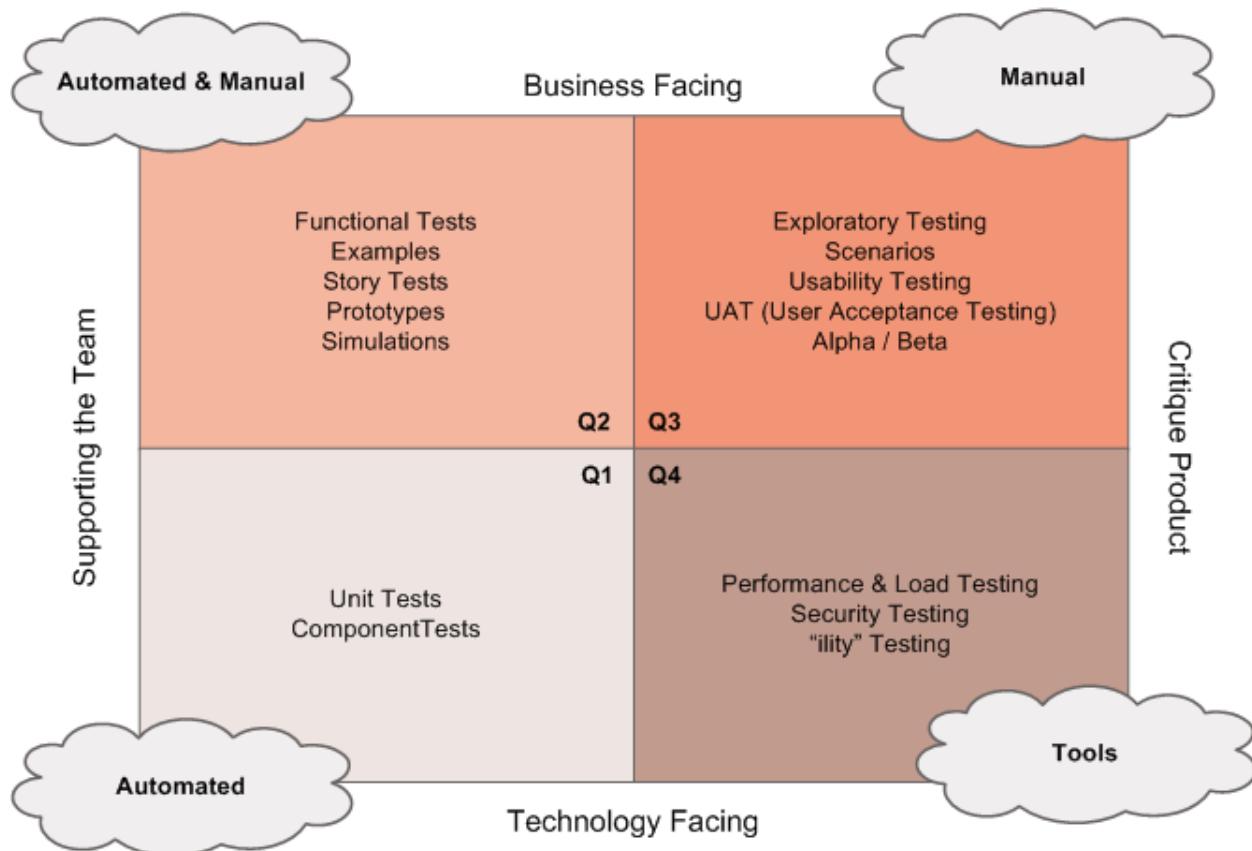
Otherwise, you can't deploy multiple times a day. But how to deal with testing, then?

Many companies still have a broad suite of manual tests to be run before delivering to production. Somehow these tests need to run every time a new release is created.

Instead of automating all your manual tests into automated UI tests, you need to rethink your testing strategy.

Lisa Crispin describes in her book Agile Testing that you can divide your tests into multiple categories.

## Agile Testing Quadrants



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

We can make four quadrants where each side of the square defines our targets with our tests.

- Business facing - the tests are more functional and often executed by end users of the system or by specialized testers that know the problem domain well.
- Supporting the Team - it helps a development team get constant feedback on the product to find bugs quickly and deliver a quality build-in product.
- Technology facing - the tests are rather technical and non-meaningful to business people. They're typical tests written and executed by the developers in a development team.
- Critique Product - tests that validate a product's workings on its functional and non-functional requirements.

Now we can place different test types we see in the other quadrants. For example, we can put Unit tests, Component tests, and System or integration tests in the first quadrant.

We can place functional tests, Story tests, prototypes, and simulations in quadrant two. These tests support the team in delivering the correct functionality and are business-facing since they're more functional.

In quadrant three, we can place tests like exploratory, usability, acceptance, etc.

We place performance, load, security, and other non-functional requirements tests in quadrant four.

Looking at these quadrants, specific tests are easy to automate or automated by nature. These tests are in quadrants 1 and 4. Tests that are automatable but mostly not automated by nature are the tests in quadrant 2. Tests that are the hardest to automate are in quadrant 3.

We also see that the tests that can't be automated or are hard to automate are tests that can be executed in an earlier phase and not after release.

We call shift-left, where we move the testing process towards the development cycle.

We need to automate as many tests as possible and test them.

A few of the principles we can use are:

- Tests should be written at the lowest level possible.
- Write once, and run anywhere, including the production system.
- The product is designed for testability.
- Test code is product code; only reliable tests survive.
- Test ownership follows product ownership.

By testing at the lowest level possible, you'll find many tests that don't require infrastructure or applications to be deployed.

We can use the pipeline to execute the tests that need an app or infrastructure. We can run scripts or use specific test tools to perform tests within the pipeline.

On many occasions, you execute these external tools from the pipeline, like Owasp ZAP, SpecFlow, or Selenium.

You can use test functionality from a platform like Azure on other occasions. For example, Availability or Load Tests executed from within the cloud platform.

When you want to write your automated tests, choose the language that resembles the language from your code.

In most cases, the application developers should also write the test, so it makes sense to use the same language. For example, write tests for your .NET application in .NET and your Angular application in Angular.

The build and release agent can handle it to execute Unit Tests or other low-level tests that don't need a deployed application or infrastructure.

When you need to do tests with a UI or other specialized functionality, you need a Test agent to run the test and report the results. Installation of the test agent then needs to be done upfront or as part of the execution of your pipeline.

## Understand Shift-left

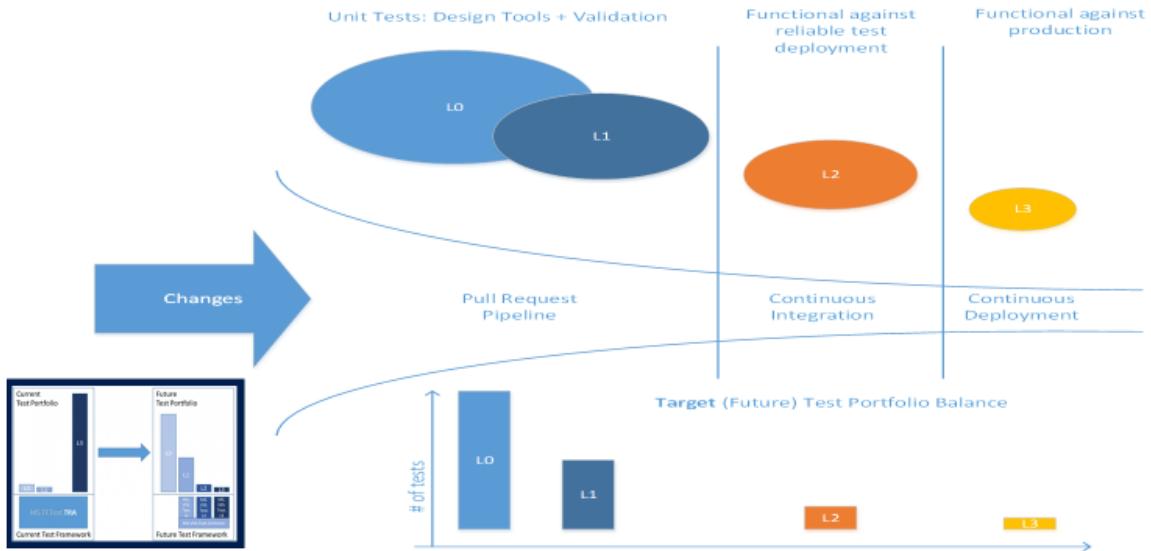
Completed 100 XP

- 4 minutes

The goal for shifting left is to move quality upstream by performing tests early in the pipeline. It represents the phrase "fail fast, fail often" combining test and process improvements reduces the time it takes for tests to be run and the impact of failures later on.

The idea is to ensure that most of the testing is complete before merging a change into the main branch.

# "Shift-Left" == Pushing Quality Upstream

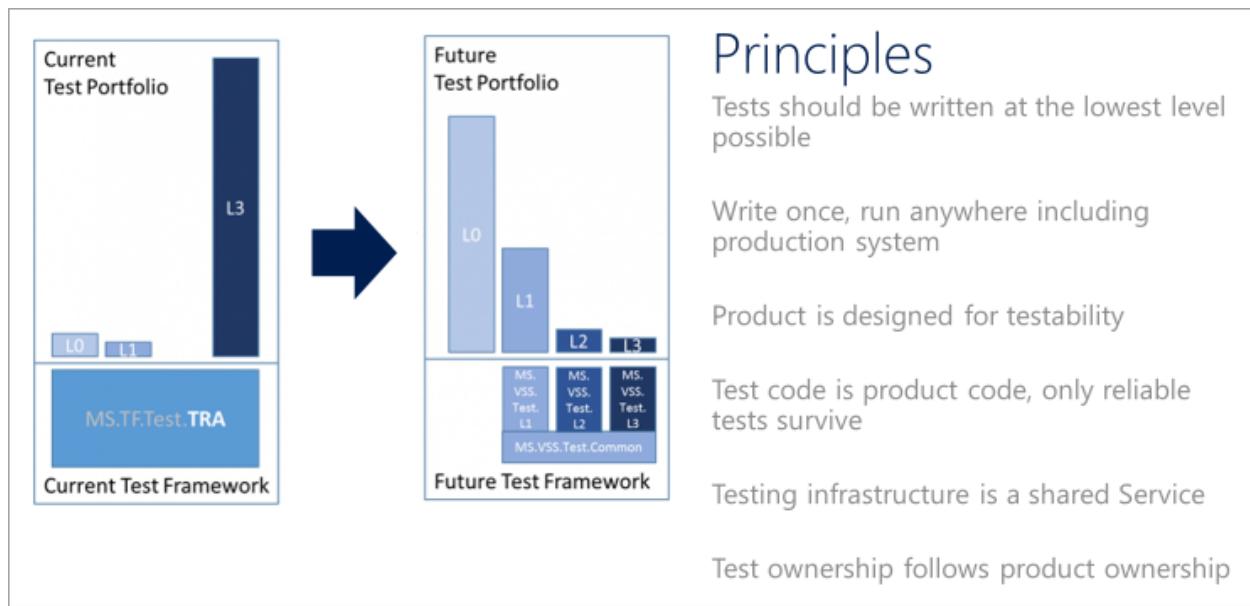


Many teams find their test takes too long to run during the development lifecycle.

As projects scale, the number and nature of tests will grow substantially, taking hours or days to run the complete test.

They get pushed further until they're run at the last possible moment, and the benefits intended to be gained from building those tests aren't realized until long after the code has been committed.

There are several essential principles that DevOps teams should adhere to in implementing any quality vision.



Other important characteristics to take into consideration:

- **Unit tests:** These tests need to be fast and reliable.
  - One team at Microsoft runs over 60,000 unit tests in parallel in less than 6 minutes, intending to get down to less than a minute.
- **Functional tests:** Must be independent.
- **Defining a test taxonomy** is an essential aspect of DevOps. The developers should understand the suitable types of tests in different scenarios.
  - **L0** tests are a broad class of fast in-memory unit tests. It's a test that depends on code in the assembly under test and nothing else.
  - **L1** tests might require assembly plus SQL or the file system.
  - **L2** tests are functional tests run against testable service deployments. It's a functional test category requiring a service deployment but may have critical service dependencies stubbed out.
  - **L3** tests are a restricted class of integration tests that run against production. They require a complete product deployment.

Check the case study in shifting left at Microsoft: [Shift left to make testing fast and reliable](#).

For more information, see:

- [Shift right to test in production](#).

## Set up and run availability tests

Completed 100 XP

- 2 minutes

After you've deployed your web app or website to any server, you can set up tests to monitor its availability and responsiveness.

It's helpful to check if your application is still running and gives a healthy response.

Some applications have specific Health endpoints that an automated process can check. The Health endpoint can be an HTTP status or a complex computation that uses and consumes crucial parts of your application.

For example, you can create a Health endpoint that queries the database. This way, you can check that your application is still accessible, but also the database connection is verified.

You can create your framework to create availability tests (ping test) or use a platform that can do it for you.

Azure has the functionality to develop Availability tests. You can use these tests in the pipeline and as release gates.

In Azure, you can set up availability tests for any HTTP or HTTPS endpoint accessible from the public internet.

You don't have to add anything to the website you're testing. It doesn't even have to be your site: you could try a REST API service you depend on.

There are two types of availability tests:

- URL ping test: a simple test that you can create in the Azure portal. You can check the URL and check the response and status code of the response.
- Multi-step web test: Several HTTP calls that are executed in sequence.

For more information, see also:

- [Creating an Application Insights Web Test and Alert Programmatically](#).
- [Monitor the availability of any website](#).

## Explore Azure Load Testing

Completed 100 XP

- 4 minutes

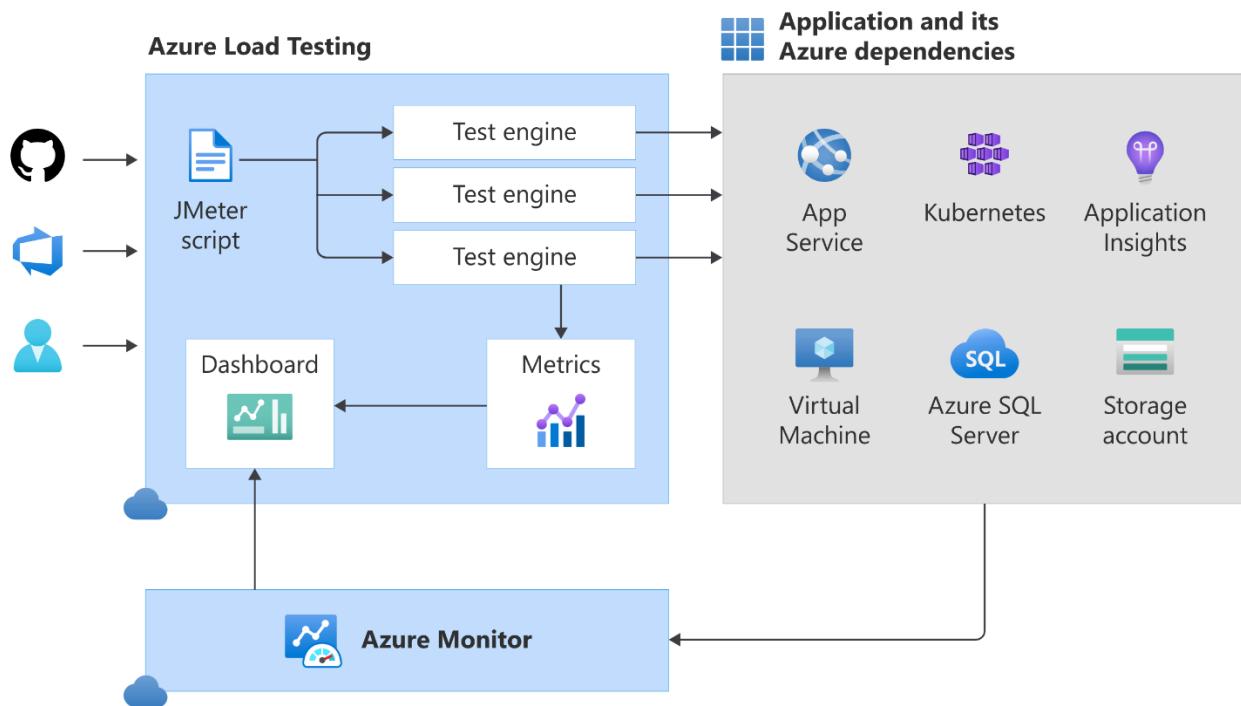
Azure Load Testing is a fully managed load-testing service that enables you to generate a high-scale load.

The service simulates your applications' traffic, helping you optimize application performance, scalability, or capacity.

You can create a load test using existing test scripts based on Apache JMeter. Azure Load Testing abstracts the infrastructure to run your JMeter script and load test your application.

Azure Load Testing collects detailed resource metrics for Azure-based applications to help you [identify performance bottlenecks](#) across your Azure application components.

You can [automate regression testing](#) by running load tests as part of your continuous integration and continuous deployment (CI/CD) workflow.



## Note

The overview image shows how Azure Load Testing uses Azure Monitor to capture metrics for app components. Learn more about the [supported Azure resource types](#).

You can automatically run a load test at the end of each sprint or in a staging environment to validate a release candidate build.

You can trigger Azure Load Testing from Azure Pipelines or GitHub Actions workflows.

Get started with [adding load testing to your Azure Pipelines CI/CD workflow](#), or use our [Azure Load Testing GitHub action](#).

For more information about the Azure Load Testing preview, see:

- [What is Azure Load Testing?](#)
- [Tutorial: Use a load test to identify performance bottlenecks.](#)
- [Tutorial: Set up automated load testing.](#)
- Learn about the [key concepts for Azure Load Testing](#).
- [Quickstart: Create and run a load test with Azure Load Testing](#).
- [Tutorial: Identify performance regressions with Azure Load Testing and GitHub Actions - Azure Load Testing](#).
- [Configure Azure Load Testing for high-scale load tests - Azure Load Testing](#).

## Set up and run functional tests

Completed 100 XP

- 60 minutes

**Estimated time:** 60 minutes.

**Lab files:** none.

### Scenario

[Selenium](#) is a portable open source software-testing framework for web applications. It can operate on almost every operating system. It supports all modern browsers and multiple languages, including .NET (C#) and Java.

This lab will teach you how to execute Selenium test cases on a C# web application as part of the Azure DevOps Release pipeline.

### Objectives

After completing this lab, you'll be able to:

- Configure a self-hosted Azure DevOps agent.
- Configure the release pipeline.
- Trigger build and release.
- Run tests in Chrome and Firefox.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft account or a Microsoft Entra account with the Contributor or the Owner role in the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](#).

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Implement Selenium tests by using a self-hosted Azure DevOps agent.
- Exercise 2: Remove the Azure lab resources.

# Explore release recommendations

## Understand the delivery cadence and three types of triggers

Completed 100 XP

- 1 minute

Release and stages make use of triggers. There are three types of triggers we recognize.

### Continuous deployment trigger

You can set up this trigger on your release pipeline. Once you do that, your release pipeline will trigger every time a build completes and creates a new release.

### Scheduled triggers

It allows you to set up a time-based manner to start a new release—for example, every night at 3:00 AM or 12:00 PM. You can have one or multiple daily schedules, but it will always run at this specific time.

### Manual trigger

With a manual trigger, a person or system triggers the release based on a specific event. When it's a person, it probably uses some UI to start a new release. When it's an automated process, some events will likely occur. You can trigger the release from another system using the automation engine, which is usually part of the release management tool.

For more information, see also:

- [Release and Stage triggers](#).

# Exercise - select your delivery and deployment cadence

Completed 100 XP

- 30 minutes

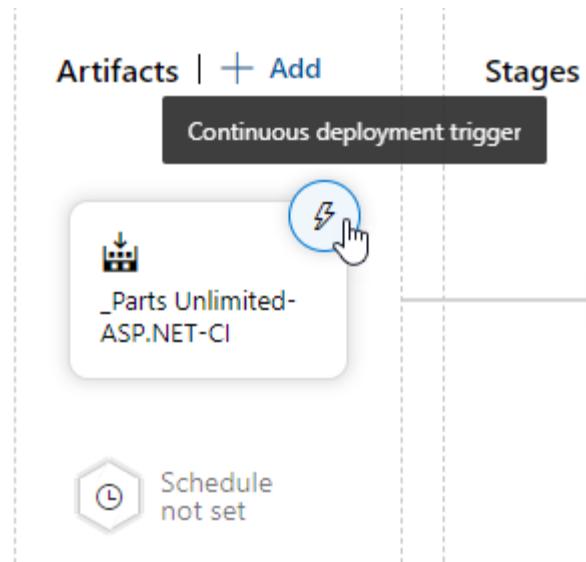
In this exercise, you'll investigate Delivery Cadence.

## Steps

Let's look at when our release pipeline is used to create deployments. Mainly, it will involve the use of triggers.

When we refer to deployment, we refer to each stage. Each stage can have its triggers that determine when the deployment occurs.

1. Click the lightning bolt on the **\_Parts Unlimited-ASP.NET-CI** artifact.



2. In the Continuous Deployment trigger pane, click the **Disabled** option to enable continuous deployment. It will then say **Enabled**.

## Continuous deployment trigger

Build: \_Parts Unlimited-ASP.NET-CI

 Enabled

Creates a release every time a new build is available.

Build branch filters 

No filters added.

 + Add | 

## Pull request trigger

Build: \_Parts Unlimited-ASP.NET-CI

 Disabled

 Enabling this will create a release every time a selected artifact is available as part of a pull request workflow

Once it's selected, every time a build completes, deployment of the release pipeline will start.

### Note

You can filter which branches affect it; for example, you could choose the main branch or a particular feature branch.

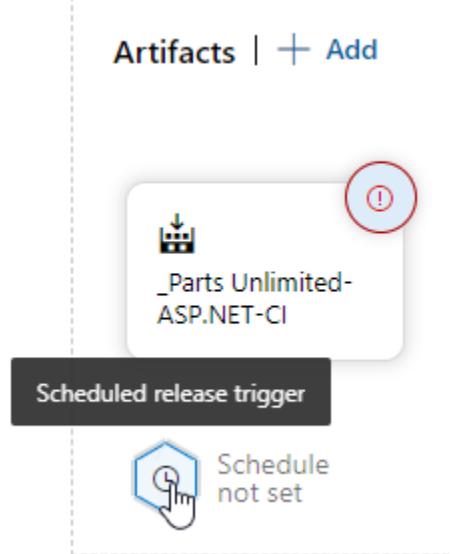
## Scheduled deployments

You might not want to have a deployment start every time a build completes.

It might be disruptive to testers downstream if it was happening too often.

Instead, it might make sense to set up a deployment schedule.

3. Click on the **Scheduled release trigger** icon to open its settings.



4. In the Scheduled release trigger pane, click the **Disabled** option to enable the scheduled release. It will then say **Enabled** and extra options will appear.

## Scheduled release trigger

Define schedules to trigger releases

Enabled

Create a new release at the specified times

Mon through Fri at 3:00 ▾

Mon  Tue  Wed  Thu  Fri  Sat  Sun

03h ▾ 00m ▾ (UTC) Coordinated Universal Time ▾

Only schedule releases if the source or pipeline has changed

+ Add a new time

You can see in the screenshot that a deployment using the release pipeline would now occur each weekday at 3 AM.

For example, it might be convenient when you share a stage with testers who work during the day.

You don't want to constantly deploy new versions to that stage while they're working. This setting would create a clean, fresh environment for them at 3 AM each weekday.

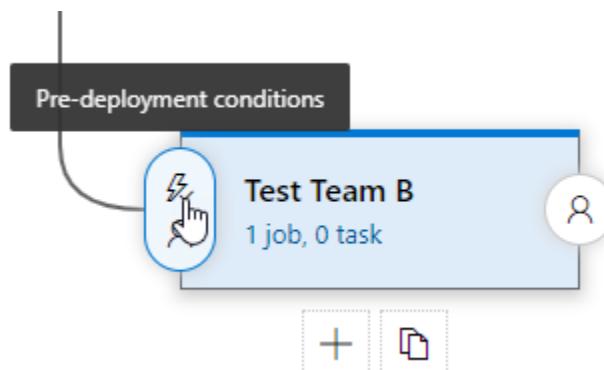
### Note

The default timezone is UTC. You can change it to suit your local timezone, as it might be easier to work with when creating schedules.

- For now, we don't need a scheduled deployment. Click the **Enabled** button again to turn off the scheduled release trigger and close the pane.

## Pre-deployment triggers

- Click the lightning bolt on the **Development** stage to open the pre-deployment conditions.



### Note

Both artifact filters and schedules can be set at the pre-deployment for each stage rather than just at the artifact configuration level.

Deployment to any stage doesn't happen automatically unless you have chosen to allow that.

## Explore release approvals

Completed 100 XP

- 4 minutes

As we've described in the introduction, Continuous Delivery is all about delivering on-demand.

But, as we discussed in the differences between release and deployment, delivery, or deployment, it's only the technical part of the Continuous Delivery process.

It's all about how you can technically install the software on an environment, but it doesn't say anything about the process that needs to be in place for a release.

Release approvals don't control *how* but control *if* you want to deliver multiple times a day.

Manual approvals also suit a significant need. Organizations that start with Continuous Delivery often lack a certain amount of trust.

They don't dare to release without manual approval. After a while, when they find that the approval doesn't add value and the release always succeeds, the manual approval is often replaced by an automatic check.

Things to consider when you're setting up a release approval are:

- What do we want to achieve with the approval? Is it an approval that we need for compliance reasons? For example, we need to adhere to the four-eyes principle to get out SOX compliance. Or is it an approval that we need to manage our dependencies? Or is it an approval that needs to be in place purely because we need a sign-off from an authority like Security Officers or Product Owners.
- Who needs to approve? We need to know who needs to approve the release. Is it a product owner, Security officer, or just someone that isn't the one that wrote the code? It's essential because the approver is part of the process. They're the ones that can delay the process if not available. So be aware of it.
- When do you want to approve? Another essential thing to consider is when to approve. It's a direct relationship with what happens after approval. Can you continue without approval? Or is everything on hold until approval is given. By using scheduled deployments, you can separate approval from deployment.

Although manual approval is a great mechanism to control the release, it isn't always helpful.

On many occasions, the check can be done at an earlier stage.

For example, it's approving a change that has been made in Source Control.

Scheduled deployments have already solved the dependency issue.

You don't have to wait for a person in the middle of the night. But there's still a manual action involved.

If you want to eliminate manual activities but still want control, you start talking about automatic approvals or release gates.

- [Release approvals and gates overview](#).

## Exercise - set up manual approvals

Completed 100 XP

- 30 minutes

In this exercise, you'll investigate Manual Approval.

### Steps

Let's now look at when our release pipeline needs manual approval before the deployment of a stage starts or manual approval that the deployment is completed as expected.

While DevOps is all about automation, manual approvals are still helpful. There are many scenarios where they're needed. For example, a product owner might want to sign off a release before it moves to production.

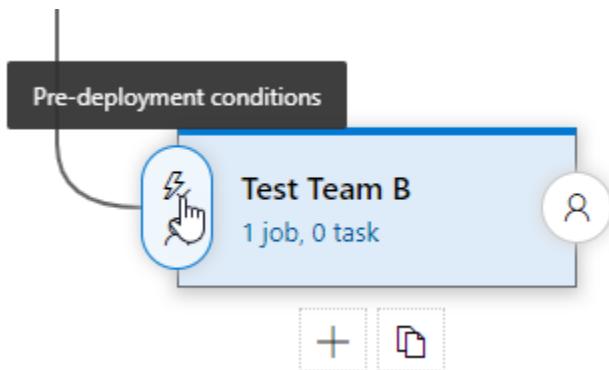
Or the scrum team wants to ensure that no new software is deployed to the test environment before someone signs off on it because they might need to find an appropriate time if it's constantly in use.

This can help to gain trust in the DevOps processes within the business.

Even if the process is later automated, people might still want manual control until they become comfortable with the processes. Explicit manual approvals can be a great way to achieve that.

Let's try one.

1. Click the pre-deployment conditions icon for the **Development** stage to open the settings.



- Click the **Disabled** button in the Pre-deployment approvals section to enable it.

**Pre-deployment approvals** Enabled

Select the users who can approve or reject deployments to this stage

Approvers

Search users and groups for approvers

Enter at least one approver.

Timeout

30 Days

Approval policies

The user requesting a release or deployment should not approve it

Skip approval if the same approver approved the previous stage

- In the **Approvers** list, find your name and select it. Then set the **Timeout** to **1 Day**.

Pre-deployment approvals

Select the users who can approve or reject deployments to this stage

Enabled

Approvers

Search users and groups for approvers

Timeout

1 Days

### Note

*Approvers is a list, not just a single value. If you add more than one person to the list, you can also choose if they need to approve in sequence or if either or both approvals are required.*

4. Take record of the approver policy options that are available:

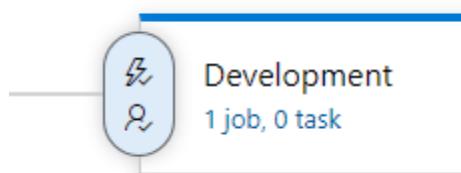
#### Approval policies

- The user requesting a release or deployment should not approve it
- Skip approval if the same approver approved the previous stage

It's prevalent not to allow a user who requests a release or deployment to approve it.

We're the only approver in this case, so we'll leave that unchecked.

5. Close the Pre-deployment conditions pane and notice a checkmark beside the person in the icon.



## Test the approval

Now it's time to see what happens when approval is required.

6. Click **Save** to save the work, and **OK** on the popup window.
7. Click **Create release** to start the release process.



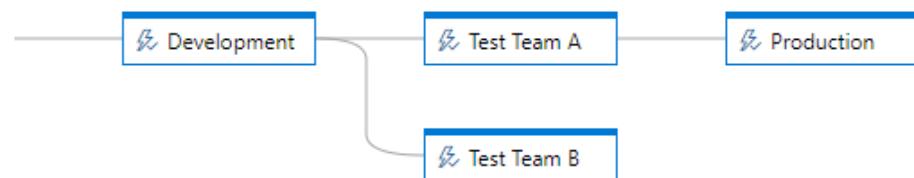
8. See the available options in the **Create a new release** pane, then click **Create**.

## Create a new release

Release to all environments

### Pipeline ▾

Click on a stage to change its trigger from automated to manual.



Stages for a trigger change from automated to manual. ⓘ

|  |  |   |
|--|--|---|
|  |  | ▼ |
|  |  |   |
|  |  |   |

### Artifacts ▾

Select the version for the artifact sources for this release

| Source alias                | Version    | ▼ |
|-----------------------------|------------|---|
| _Parts Unlimited-ASP.NET-CI | 20190901.2 | ▼ |

### Release description

|  |
|--|
|  |
|  |
|  |

**Create**

Cancel

9. In the upper left of the screen, you can see that a release has been created.

All pipelines >  Release to all environments

 Release **Release-1** has been created

Pipeline

Tasks ▾

Variables

Retention

Options

History

10. An email should have been received at this point, indicating that approval is required.



Azure DevOps

Release to all environments > Release-1

## Deployment to Development is waiting for your approval

 Parts Unlimited-ASP.NET-CI / 20190901.2

Requested for

[View approval](#)

### Summary

**Release description** ("None")

**Deployment trigger** automated: after release creation

**Requested for**

**Attempt** 1

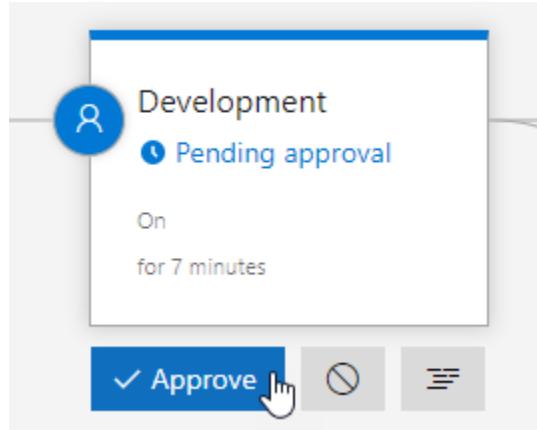
At this point, you could click the link in the email, but instead, we'll navigate within Azure DevOps to see what's needed.

11. Click on the **Release 1 Created** link (or whatever number it is for you) in the area we looked at in Step 9. We're then taken to a screen that shows the status of the release.

The screenshot shows the Azure DevOps Release pipeline interface. At the top, there's a breadcrumb navigation: 'Release to all environments > Release-1'. Below it, a navigation bar includes 'Pipeline' (which is underlined), 'Variables', 'History', and buttons for '+ Deploy', 'Cancel', and 'Approve multiple'. The main area is divided into two sections: 'Release' on the left and 'Stages' on the right. The 'Release' section contains information about a 'Manually triggered' release by a user on 01/09/2019, 16:40. It also lists an artifact named '\_Parts Unlimited-ASP.N... 20190901.2'. The 'Stages' section shows a single stage named 'Development' which is currently 'Pending approval'. A blue circle highlights the 'Approve' icon next to the stage name. Below the stage, it says 'On for 4 minutes'.

You can see that a release has been manually triggered and that the Development stage is waiting for approval. As an approver, you can now do that approval.

12. Hover over the **Development** stage and click the **Approve** icon that appears.



### Note

Options to cancel the deployment or to view the logs are also provided at this point.

13. In the Development approvals window, add a comment and click **Approve**.

Development

Pre-deployment conditions • Pending approval

Approvers | [View logs](#)

⌚ Approval pending for 8 minutes  
Waiting for all approvers to approve in sequence .

⌚ Pending for 8 minutes

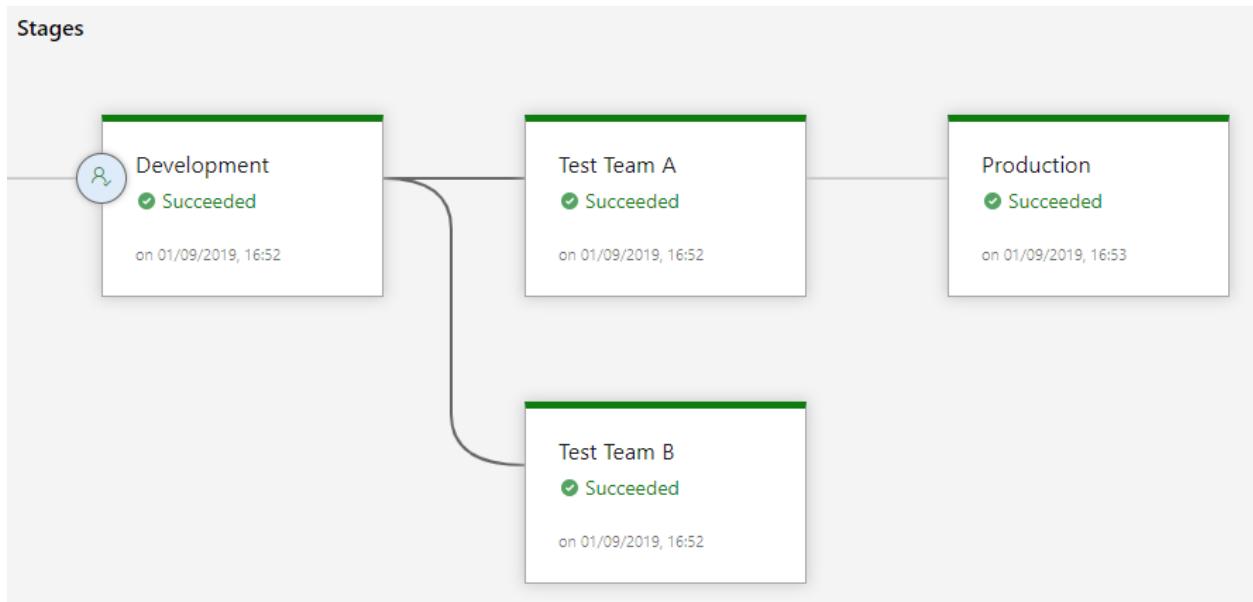
**Comment**

Great work people

Defer deployment for later

**Approve** **Reject**

The deployment stage will then continue. Watch as each stage proceeds and succeeds.



## Explore release gates

Completed 100 XP

- 4 minutes

Release gates give you more control over the start and completion of the deployment pipeline.

They're often set up as pre-deployment and post-deployment conditions.

In many organizations, there are so-called dependency meetings.

It's a planning session where the release schedule of dependent components is discussed.

Think of downtime of a database server or an update of an API.

It takes much time and effort, and the only thing needed is a signal if the release can continue.

Instead of having this meeting, you can create a mechanism where people press a button on a form when the release can't advance.

When the release starts, it checks the state of the gate by calling an API. If the "gate" is open, we can continue. Otherwise, we'll stop the release.

By using scripts and APIs, you can create your release gates instead of manual approval. Or at least extending your manual approval.

Other scenarios for automatic approvals are, for example.

- Incident and issues management. Ensure the required status for work items, incidents, and issues. For example, ensure that deployment only occurs if no bugs exist.
- Notify users such as legal approval departments, auditors, or IT managers about a deployment by integrating with approval collaboration systems such as Microsoft Teams or Slack and waiting for the approval to complete.
- Quality validation. Query metrics from tests on the build artifacts such as pass rate or code coverage and only deploy within required thresholds.
- Security scan on artifacts. Ensure security scans such as anti-virus checking, code signing, and policy checking for build artifacts have been completed. A gate might start the scan and wait for it to finish or check for completion.
- User experience relative to baseline. Using product telemetry, ensure the user experience hasn't regressed from the baseline state. The experience level before the deployment could be considered a baseline.
- Change management. Wait for change management procedures in a system such as ServiceNow complete before the deployment occurs.
- Infrastructure health. Execute monitoring and validate the infrastructure against compliance rules after deployment or wait for proper resource use and a positive security report.

In short, approvals and gates give you more control over the start and completion of the deployment pipeline.

They can usually be set up as pre-deployment and post-deployment conditions, including waiting for users to approve or reject deployments manually and checking with other automated systems until specific requirements are verified.

Also, you can configure a manual intervention to pause the deployment pipeline and prompt users to carry out manual tasks, then resume or reject the deployment.

To find out more about Release Approvals and Gates, check these documents.

- [Release approvals and gates overview.](#)
- [Release Gates.](#)

# Use release gates to protect quality

Completed 100 XP

- 3 minutes

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?

A quality gate is located before a stage that is dependent on the outcome of a previous stage. A quality gate was typically something that a QA department monitored in the past.

They had several documents or guidelines, and they verified if the software was of a good enough quality to move on to the next stage.

When we think about Continuous Delivery, all manual processes are a potential bottleneck.

We need to reconsider the notion of quality gates and see how we can automate these checks as part of our release pipeline.

By using automatic approval with a release gate, you can automate the approval and validate your company's policy before moving on.

Many quality gates can be considered.

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

Defining quality gates improves the release process, and you should always consider adding them.

# Control Deployments using Release Gates

Completed 100 XP

- 75 minutes

**Estimated time:** 75 minutes.

**Lab files:** none.

## Scenario

This lab covers the configuration of the deployment gates and details how to use them to control the execution of Azure Pipelines. To illustrate their implementation, you'll configure a release definition with two environments for an Azure Web App. You'll deploy to the Canary environment only when there are no blocking bugs for the app and mark the Canary environment complete only when there are no active alerts in Application Insights of Azure Monitor.

A release pipeline specifies the end-to-end release process for an application to be deployed across various environments. Deployments to each environment are fully automated by using jobs and tasks. Ideally, you don't want new updates to the applications to be simultaneously exposed to all users. It's a best practice to expose updates in a phased manner, that is, expose them to a subset of users, monitor their usage, and expose them to other users based on the experience of the initial set of users.

Approvals and gates enable you to control the start and completion of the deployments in a release. You can wait for users to approve or reject deployments with approvals manually. Using release gates, you can specify application health criteria to be met before the release is promoted to the following environment. Before or after any environment deployment, all the specified gates are automatically evaluated until they pass or reach your defined timeout period and fail.

Gates can be added to an environment in the release definition from the pre-deployment conditions or the post-deployment conditions panel. Multiple gates can be added to the environment conditions to ensure all the inputs are successful for the release.

As an example:

- Pre-deployment gates ensure no active issues in the work item or problem management system before deploying a build to an environment.
- Post-deployment gates ensure no incidents from the app's monitoring or incident management system after being deployed before promoting the release to the following environment.

There are four types of gates included by default in every account.

- Invoke Azure Function: Trigger the execution of an Azure Function and ensures a successful completion.
- Query Azure Monitor alerts: Observe the configured Azure Monitor alert rules for active alerts.
- Invoke REST API: Make a call to a REST API and continues if it returns a successful response.
- Query work items: Ensure the number of matching work items returned from a query is within a threshold.

## Objectives

After completing this lab, you'll be able to:

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

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- Identify an existing Azure subscription or create a new one.

- Verify that you have a Microsoft or Microsoft Entra account with the Contributor or the Owner role in the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](#).

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Configure the build pipeline.
- Exercise 2: Configure the release pipeline.
- Exercise 3: Configure release gates.
- Exercise 4: Test release gates.
- Exercise 5: Remove the Azure lab resources.

## Create a release pipeline

### Describe Azure DevOps release pipeline capabilities

Completed 100 XP

- 3 minutes

Azure DevOps has extended support for pipelines as code (also called YAML pipelines) for continuous deployment and started introducing various release management capabilities into pipelines as code.

The existing UI-based release management solution in Azure DevOps is referred to as classic release.

You'll find a list of capabilities and availability in YAML pipelines vs. classic build and release pipelines in the following table.

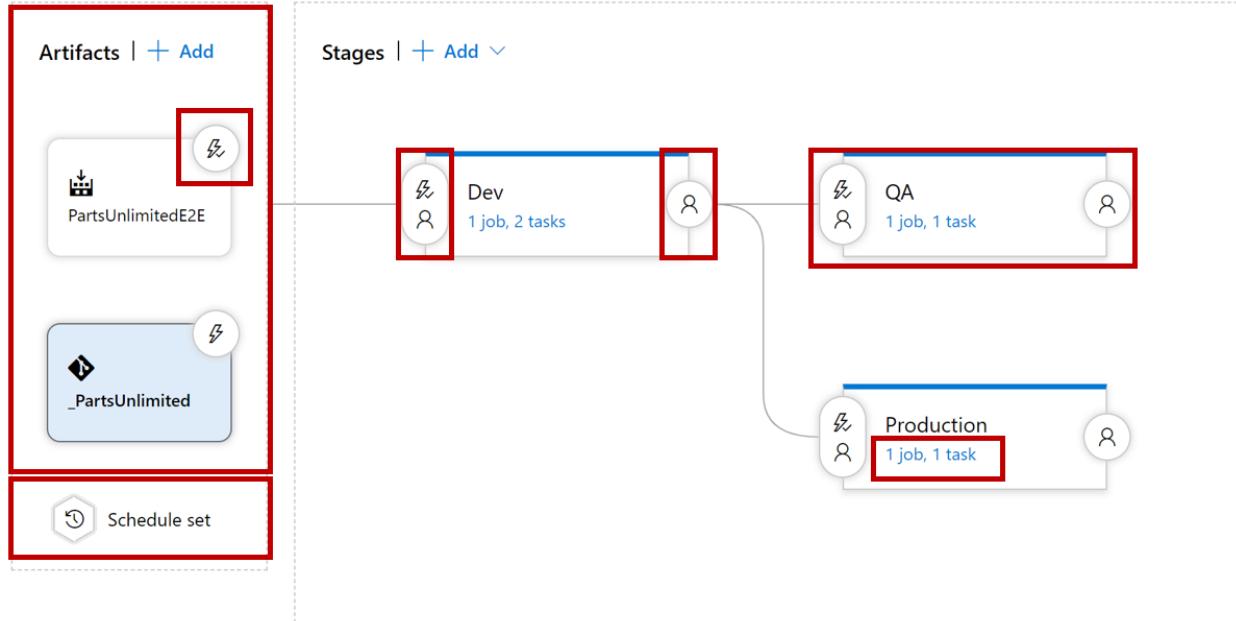
| Feature               | YAML | Classic Build | Classic Release | Notes   |
|-----------------------|------|---------------|-----------------|---|
| Agents                | Yes  | Yes           | Yes             | Specifies a required resource on which the pipeline runs.   |
| Approvals             | Yes  | No            | Yes             | Defines a set of validations required before completing a deployment stage.   |
| Artifacts             | Yes  | Yes           | Yes             | Supports publishing or consuming different package types.   |
| Caching               | Yes  | Yes           | No              | Reduces build time by allowing outputs or downloaded dependencies from one run to be reused in later runs. In Preview, available with Azure Pipelines only. |
| Conditions            | Yes  | Yes           | Yes             | Specifies conditions to be met before running a job.  |
| Container jobs        | Yes  | No            | No              | Specifies jobs to run in a container.   |
| Demands               | Yes  | Yes           | Yes             | Ensures pipeline requirements are met before running a pipeline stage. Requires self-hosted agents.   |
| Dependencies          | Yes  | Yes           | Yes             | Specifies a requirement that must be met to run the next job or stage.  |
| Deployment groups     | Yes  | No            | Yes             | Defines a logical set of deployment target machines.  |
| Deployment group jobs | No   | No            | Yes             | Specifies a job to release to a deployment group.   |
| Deployment jobs       | Yes  | No            | No              | Defines the deployment steps. Requires Multi-stage pipelines experience.  |
|                       |      |               |                 |   |
| Environment           | Yes  | No            | No              | Represents a collection of resources targeted for deployment. Available with Azure Pipelines only.  |
| Gates                 | No   | No            | Yes             | Supports automatic collection and evaluation of external health signals before completing a release stage. Available with Azure Pipelines only.             |
| Jobs                  | Yes  | Yes           | Yes             | Defines the execution sequence of a set of steps.   |
| Service connections   | Yes  | Yes           | Yes             | Enables a connection to a remote service that is required to execute tasks in a job.  |
| Service containers    | Yes  | No            | No              | Enables you to manage the lifecycle of a containerized service.   |
| Stages                | Yes  | No            | Yes             | Organizes jobs within a pipeline.   |
| Task groups           | No   | Yes           | Yes             | Encapsulates a sequence of tasks into a single reusable task. If using YAML, see templates.   |
| Tasks                 | Yes  | Yes           | Yes             | Defines the building blocks that make up a pipeline.  |
| Templates             | Yes  | No            | No              | Defines reusable content, logic, and parameters.  |
| Triggers              | Yes  | Yes           | Yes             | Defines the event that causes a pipeline to run.  |
| Variables             | Yes  | Yes           | Yes             | Represents a value to be replaced by data to pass to the pipeline.  |
| Variable groups       | Yes  | Yes           | Yes             | Use to store values that you want to control and make available across multiple pipelines.  |

# Explore release pipelines

Completed 100 XP

- 3 minutes

A release pipeline takes artifacts and releases them through stages and finally into production.



Let us quickly walk through all the components step by step.

The first component in a release pipeline is an artifact:

- Artifacts can come from different sources.
- The most common source is a package from a build pipeline.
- Another commonly seen artifact source is, for example, source control.

Furthermore, a release pipeline has a trigger: the mechanism that starts a new release.

A trigger can be:

- A manual trigger, where people start to release by hand.
- A scheduled trigger, where a release is triggered based on a specific time.
- A continuous deployment trigger, where another event triggers a release. For example, a completed build.

Another vital component of a release pipeline is stages or sometimes called environments. It's where the artifact will be eventually installed. For example, the artifact contains the compiled website installed on the webserver or somewhere in the cloud. You can have many stages (environments); part of the release strategy is finding the appropriate combination of stages.

Another component of a release pipeline is approval.

People often want to sign a release before installing it in the environment.

In more mature organizations, this manual approval process can be replaced by an automatic process that checks the quality before the components move on to the next stage.

Finally, we have the tasks within the various stages. The tasks are the steps that need to be executed to install, configure, and validate the installed artifact.

In this part of the module, we'll detail all the release pipeline components and talk about what to consider for each element.

The components that make up the release pipeline or process are used to create a release. There's a difference between a release and the release pipeline or process. The release pipeline is the blueprint through which releases are done. We'll cover more of it when discussing the quality of releases and releases processes.

See also [Release pipelines](#).

## Explore artifact sources

Completed 100 XP

- 4 minutes

What is an artifact? An artifact is a deployable component of your application. These components can then be deployed to one or more environments.

In general, the idea about build and release pipelines and Continuous Delivery is to build once and deploy many times.

It means that an artifact will be deployed to multiple environments. The artifact should be a stable package if you want to achieve it.

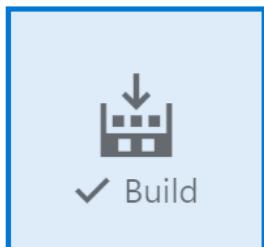
The configuration is the only thing you want to change when deploying an artifact to a new environment.

The contents of the package should never change. It's what we call [immutability](#). We should be 100% sure that the package that we build, the artifact, remains unchanged.

How do we get an artifact? There are different ways to create and retrieve artifacts, and not every method is appropriate for every situation.

## Add an artifact

Source type



Azure Repos ...



Github



TFVC



Azure Artifacts



Azure Contai...



Docker Hub



Jenkins

The most common way to get an artifact within the release pipeline is to use a build artifact.

The build pipeline compiles, tests, and eventually produces an immutable package stored in a secure place (storage account, database, and so on).

The release pipeline then uses a secure connection to this secured place to get the build artifact and do extra actions to deploy it to an environment.

The significant advantage of using a build artifact is that the build produces a versioned artifact.

The artifact is linked to the build and gives us automatic traceability. We can always find the sources that made this artifact. Another possible artifact source is version control.

We can directly link our version control to our release pipeline.

The release is related to a specific commit in our version control system. With that, we can also see which version of a file or script is eventually installed. In this case, the version doesn't come from the build but from version control.

Consideration for choosing a version control artifact instead of a build artifact can be that you only want to deploy one specific file. If you don't need to run more actions before using this file in your release pipeline, creating a versioned package (build artifact) containing only one file doesn't make sense.

Helper scripts that do actions to support the release process (clean up, rename, string actions) are typically good candidates to get from version control.

Another possibility of an artifact source can be a network share containing a set of files. However, you should be aware of the possible risk. The risk is that you aren't 100% sure that the package you're going to deploy is the same package that was put on the network share. If other people can also access the network share, the package might be compromised. Therefore, this option won't be sufficient to prove integrity in a regulated environment (banks, insurance companies).

Finally, container registries are a rising star regarding artifact sources. Container registries are versioned repositories where container artifacts are stored. Pushing a versioned container to the content repository and consuming that same version within the release pipeline has more or less the same advantages as using a build artifact stored in a safe location.

## Choose the appropriate artifact source

Completed 100 XP

- 3 minutes

When you use a release pipeline to deploy your packages to production, you need traceability.

That means **you want to know where the package that you're deploying originates from.**

It's essential to understand that the sources that you built and checked into your version control are precisely the same as the sources that you're going to deploy to the various environments that are going to be used by your customers.

Primarily when you work in a regulated environment like a bank or an insurance company, auditors ask you to provide traceability to sources that you deployed to prove the integrity of the package.

Another crucial aspect of your artifacts is auditability. You want to know who changed that line of code and who triggered the build that produces the artifact deployed.

A proper mechanism to make sure you can provide the correct traceability and auditability is using immutable packages.

It isn't something that you can buy, but something that you need to implement yourself.

Using a build pipeline that produces a package stored in a location that humans can't access, you ensure the sources are unchanged throughout the whole-release process. It's an essential concept of release pipelines.

You identify an immutable package by giving it a version so that you can refer to it at a later stage. Versioning strategy is a complex concept and isn't in the scope of this module.

Still, having a unique identification number or label attached to the package and ensuring that this number or label cannot be changed or modified afterward ensures traceability and auditability from source code to production.

Read more about [Semantic Versioning](#).

Choosing the right artifact source is tightly related to the requirements you have about traceability and auditability.

If you need an immutable package (containing multiple files) that can never be changed and be traced, a build artifact is the best choice.

If it's one file, you can directly link to source control.

You can also point at a disk or network share, but it implies some risk-concerning auditability and immutability. Can you ensure the package never changed?

See also [Release artifacts and artifact sources](#).

## Exercise - select an artifact source

Completed 100 XP

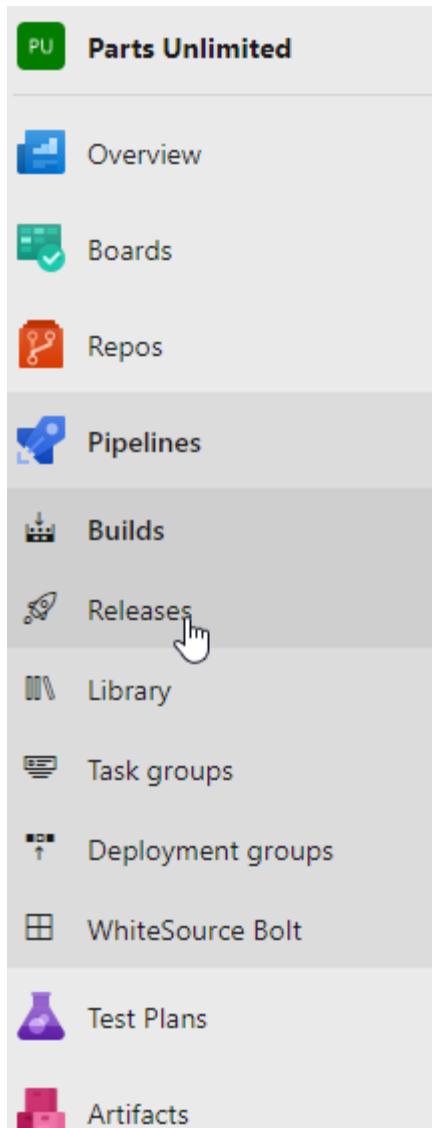
- 30 minutes

In this exercise, you'll investigate Artifact Sources.

# Steps

Let's look at how to work with one or more artifact sources in the release pipeline.

1. In the Azure DevOps environment, open the **Parts Unlimited** project, then from the main menu, select **Pipelines**, then select **Releases**.



2. In the main screen area, select **New pipeline**.



## No release pipelines found

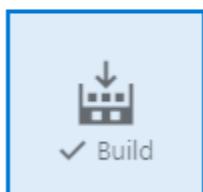
Automate your release process in a few easy steps with a new pipeline

New pipeline

3. In the **Select a template** pane, see the available templates, but then select the **Empty job** option at the top. It's because we're going to focus on selecting an artifact source.
4. In the **Artifacts** section, select **+Add an artifact**.
5. See the available options in the **Add an artifact** pane, and select the option to see **more artifact types**, so that you can see all the available artifact types:

## Add an artifact

### Source type



Show less ^

While we're in this section, let's briefly look at the available options.

6. Select **Build** and see the parameters required. This option is used to retrieve artifacts from an Azure DevOps Build pipeline. Using it requires a project name and a build pipeline name. (Projects can have multiple build pipelines). It's the option that we'll use shortly.

Project \* (i)

Source (build pipeline) \* (i)

(i) This setting is required.

7. Select **Azure Repository** and see the parameters required. It requires a project name and asks you to select the source repository.

Project \* !

! This setting is required.

Source (repository) \* !

! This setting is required.

8. Select **GitHub** and see the parameters required. The **Service** is a connection to the GitHub repository. It can be authorized by either OAuth or by using a GitHub personal access token. You also need to select the source repository.

Service \* | Manage ↗

! This setting is required.

Source (repository) \* !

! This setting is required.

9. Select **TFVC** and see the parameters required. It also requires a project name and asks you to select the source repository.

Project \* !

! This setting is required.

Source (repository) \* !

! This setting is required.

## Note

*A release pipeline can have more than one set of artifacts as input. A typical example is when you also need to consume a package from a feed and your project source.*

10. Select **Azure Artifacts** and see the parameters required. It requires you to identify the feed, package type, and package.

Feed \*

! This setting is required.

Package type

NuGet

Package \* ⓘ

! This setting is required.

11. Select **GitHub Release** and see the parameters required. It requires a service connection and the source repository.

Service connection \* | Manage ↗

! This setting is required.

Source (repository) \* ⓘ

! This setting is required.

### Note

*We'll discuss service connections later in the module.*

12. Select **Azure Container Registry** and see the parameters required. Again, it requires a secure service connection, and the Azure Resource Group details that the container registry is located. It allows you to provide all your Docker containers directly into your release pipeline.

Service connection \* | Manage ↗

! This setting is required.

Resource Group \* ⓘ

! This setting is required.

Azure Container Registry \* ⓘ

! This setting is required.

Repository \* ⓘ

! This setting is required.

13. Select **Docker Hub** and see the parameters required. This option would be helpful if your containers are stored in Docker Hub rather than in an Azure Container Registry. After choosing a secure service connection, you need to select the namespace and the repository.

Service connection \* | Manage ↗

! This setting is required.

Namespaces \* ⓘ

! This setting is required.

Repository \* ⓘ

! This setting is required.

14. Finally, select **Jenkins** and see the parameters required. You don't need to get all your artifacts from Azure. You can retrieve them from a Jenkins build. So, if you have a

Jenkins Server in your infrastructure, you can use the build artifacts from there directly in your Azure Pipelines.

Service connection \* | [Manage](#) 

! This setting is required.

Jenkins Job \* 

! This setting is required.

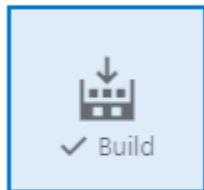
## Configuring the build artifact

Let's return to adding our Build output as the artifact source.

15. Select the **Build** source type again. See that the Project should show the current project. From the **Source (build pipeline)** drop-down list, select **Parts Unlimited-ASP.NET-CI**. Take a record of the default values for the other options, and then select **Add**.

## Add an artifact

Source type



[5 more artifact types ▾](#)

Project \* [\(i\)](#)

Parts Unlimited

Source (build pipeline) \* [\(i\)](#)

Parts Unlimited-ASP.NET-CI

Default version \* [\(i\)](#)

Latest

Source alias \* [\(i\)](#)

\_Parts Unlimited-ASP.NET-CI

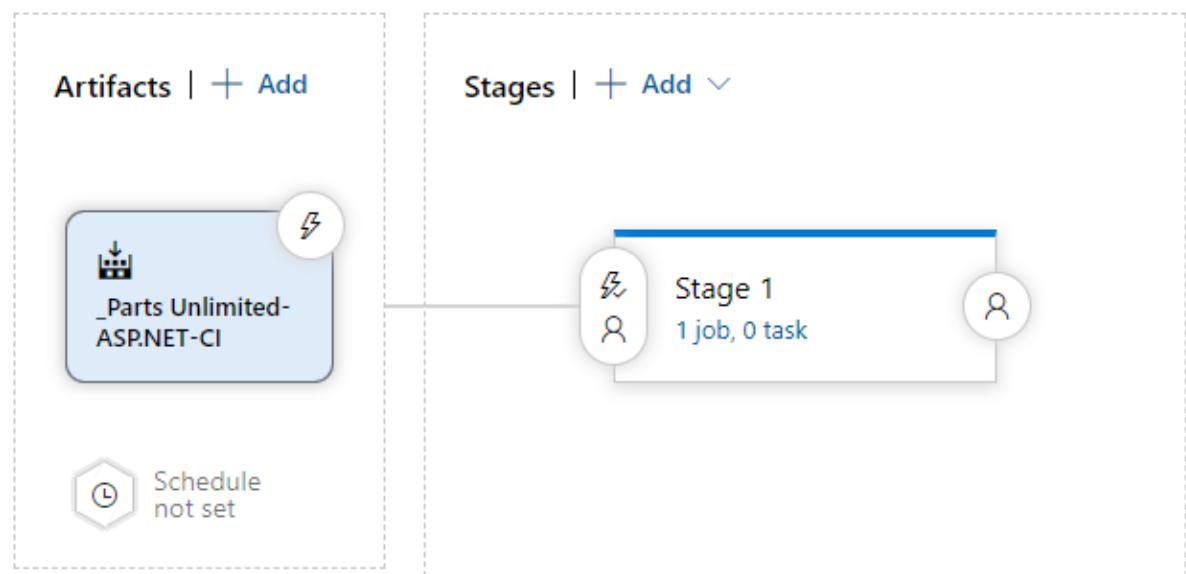
[\(i\)](#) The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **Parts Unlimited-ASP.NET-CI** published the following artifacts: **drop**.

**Add**

We've now added the artifacts that we'll need for later walkthroughs.

## All pipelines > New release pipeline

Pipeline Tasks Variables Retention Options History



16. To save the work, select **Save**, then in the Save dialog box, select **OK**.

## Examine considerations for deployment to stages

Completed 100 XP

- 3 minutes

When you have a clear view of the different stages you'll deploy, you need to think about when you want to deploy to these stages.

As we mentioned in the introduction, Continuous Delivery is about deploying multiple times a day and can deploy on-demand.

When we define our cadence, questions that we should ask ourselves are:

- Do we want to deploy our application?
- Do we want to deploy multiple times a day?

- Can we deploy to a stage? Is it used?

For example, a tester testing an application during the day might not want to deploy a new version of the app during the test phase.

Another example is when your application incurs downtime, you don't want to deploy when users use the application.

The frequency of deployment, or cadence, differs from stage to stage.

A typical scenario we often see is continuous deployment during the development stage.

Every new change ends up there once it's completed and builds.

Deploying to the next phase doesn't always occur multiple times but only at night.

When designing your release strategy, choose your triggers carefully and consider the required release cadence.

Some things we need to take into consideration are:

- What is your target environment?
- Does one team use it, or do multiple teams use it?
  - If a single team uses it, you can deploy it frequently. Otherwise, it would be best if you were a bit more careful.
- 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 affected?

## Exercise - set up stages

Completed 100 XP

- 30 minutes

In this demonstration, you'll investigate Stages.

## Steps

Look at the other section in our release pipeline: Stages.

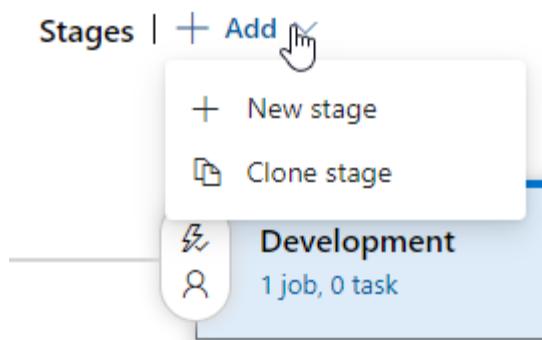
1. Click on **Stage 1**, and in the Stage properties pane, set **Stage name** to **Development** and close the pane.

The screenshot shows the 'Stage' properties pane for the 'Development' stage. At the top, there are buttons for Delete, Move, and more. Below that, the stage name is listed as 'Development'. Under 'Stage name', the value 'Development' is shown in a highlighted input field. Under 'Stage owner', there is a search bar with the placeholder 'Search users and groups' and a red error message: 'Enter a valid identity.'.

### Note

*Stages can be based on templates. For example, you might deploy a web application using node.js or Python. For this walkthrough that won't matter because we're focusing on defining a strategy.*

2. To add a second stage, click **+Add** in the Stages section and note the available options. You have a choice to create a new stage or clone an existing stage. Cloning a stage can help minimize the number of parameters that need to be configured. But for now, click **New stage**.



3. When the **Select a template** pane appears, scroll down to see the available templates. We don't need any of these, so click **Empty job** at the top, then in the Stage properties pane, set **Stage name** to **Test**, and then close the pane.

Select a template

Or start with an  [Empty job](#)

Search

---

Featured

**Azure App Service deployment**  
Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.

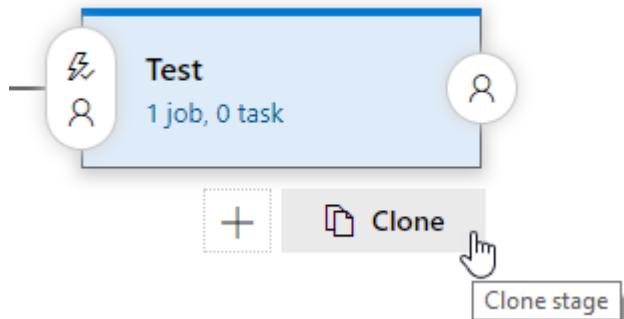
**Deploy a Java app to Azure App Service**  
Deploy a Java application to an Azure Web App.

**Deploy a Node.js app to Azure App Service**  
Deploy a Node.js application to an Azure Web App.

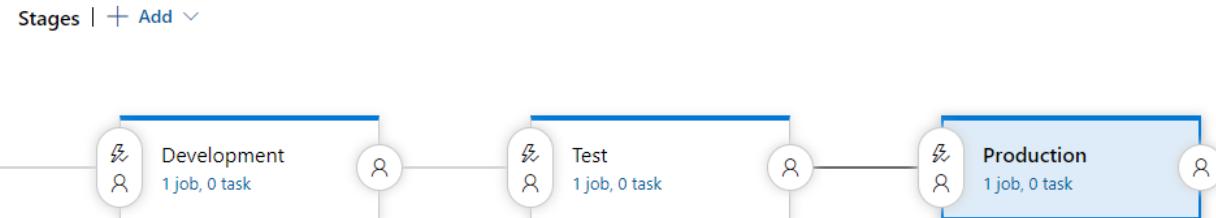
**Deploy a PHP app to Azure App Service and Azure Database for MySQL**  
Deploy a PHP application to an Azure Web App and database to Azure Database for MySQL.

**Deploy a Python app to Azure App Service and**

4. Hover over the **Test** stage and notice that two icons appear below. These are the same options that were available in the menu drop-down that we used before. Click the **Clone** icon to clone the stage to a new stage.



- Click on the **Copy of Test** stage, and in the stage properties pane, set **Stage name** to **Production** and close the pane.



We've now defined a traditional deployment strategy. Each stage contains a set of tasks, and we'll look at those tasks later in the course.

#### Note

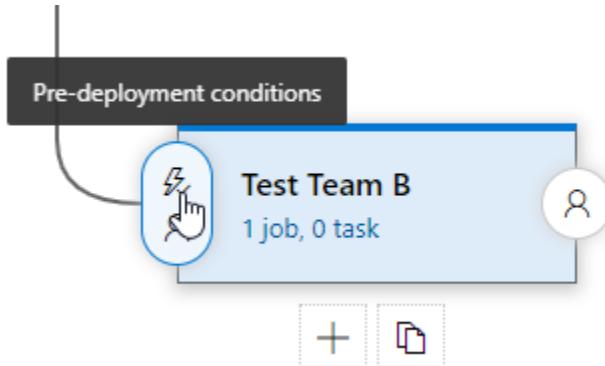
The same artifact sources move through each of the stages.

The lightning bolt icon on each stage shows that we can set a trigger as a pre-deployment condition. The person icon on both ends of a stage shows that we can have pre and post-deployment approvers.

## Concurrent stages

You'll notice that now, we have all the stages one after each other in a sequence. It's also possible to have concurrent stages. Let's see an example.

- Click the **Test** stage, and on the stage properties pane, set the **Stage name** to **Test Team A** and close the pane.
- Hover over the **Test Team A** stage and click the **Clone** icon that appears to create a new cloned stage.
- Click the **Copy of Test Team A** stage, and on the stage properties pane, set **Stage name** to **Test Team B** and close the pane.
- Click the **Pre-deployment conditions** icon (that is, the lightning bolt) on **Test Team B** to open the pre-deployment settings.



10. In the Pre-deployment conditions pane, the stage can be triggered in three different ways:

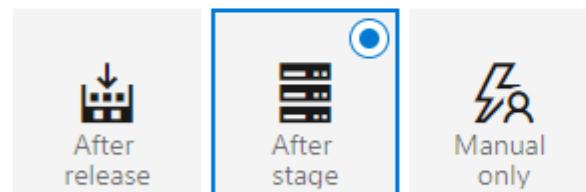
### Pre-deployment conditions

Test Team B

#### Triggers ^

Define the trigger that will start deployment to this stage

Select trigger ⓘ



Stages ⓘ

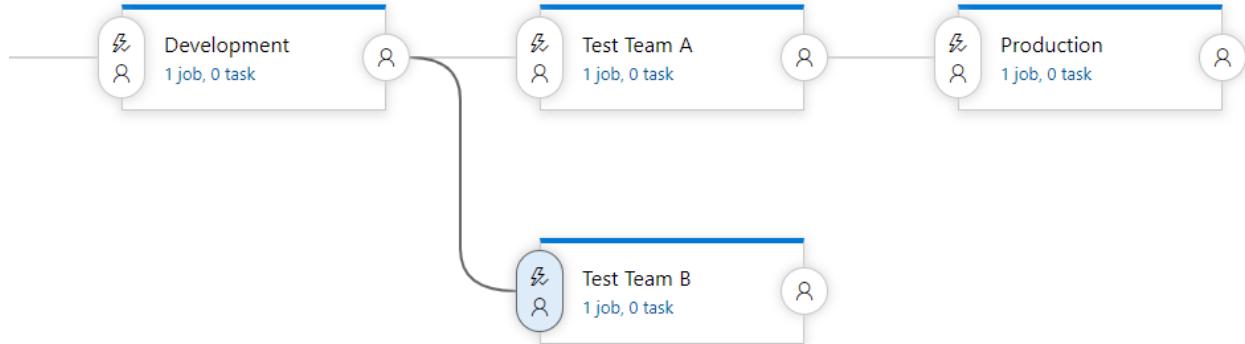
Test Team A

The stage can immediately follow Release. (That is how the Development stage is currently configured). It can require manual triggering. Or, more commonly, it can follow another stage. Now, it's following **Test Team A**, but that's not what we want.

11. Choose Development from the **Stages** drop-down list\*\*, \*\* uncheck **Test Team A**, and then close the pane.

We now have two concurrent Test stages.

Stages | + Add ▾

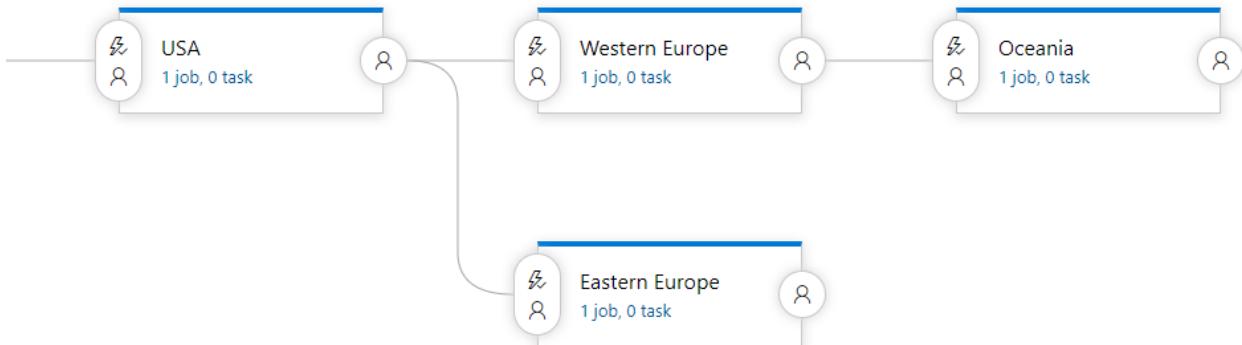


## Stage versus Environment

You may have wondered why these items are called **Stages** and not **Environments**.

In the current configuration, we're using them for different environments. But it's not always the case. Here's a deployment strategy based upon regions instead:

Stages | + Add ▾



Azure Pipelines are configurable and support a wide variety of deployment strategies. The name **Stages** is a better fit than **Environment** even though the stages can be used for environments.

Let's give the pipeline a better name and save the work.

12. At the top of the screen, hover over the **New release pipeline** name and click it to edit the name when a pencil appears. Type **Release to all environments** as the name and hit enter or click elsewhere on the screen.



13. For now, save the environment-based release pipeline you've created by clicking **Save**. Then, **click OK** in the Save dialog box.

## Explore build and release tasks

Completed 100 XP

- 4 minutes

A build and release platform requires executing any number of repeatable actions during the build process. Tasks are units of executable code used to do selected actions in a specified order.

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

Add steps to specify what you want to build. The tests you want to run and all the other steps needed to complete the build process.

There are steps for building, testing, running utilities, packaging, and deploying.

If a task isn't available, you can find numerous community tasks in the marketplace.

Jenkins, Azure DevOps, and Atlassian have an extensive marketplace where other tasks can be found.

## Links

For more information, see also:

- [Task types & usage](#)
- [Tasks for Azure](#)
- [Atlassian marketplace](#)
- [Jenkins Plugins](#)

# Explore custom build and release tasks

Completed 100 XP

- 3 minutes

Instead of using out-of-the-box tasks, a command line, or a shell script, you can also use your custom build and release task.

By creating your tasks, the tasks are available publicly or privately to everyone you share them with.

Creating your task has significant advantages.

- You get access to variables that are otherwise not accessible.
- You can use and reuse a secure endpoint to a target server.
- You can safely and efficiently distribute across your whole organization.
- Users don't see implementation details.

For more information, see [Add a build or release task](#).

# Explore release jobs

Completed 100 XP

- 4 minutes

You can organize your build or release pipeline into jobs. Every build or deployment pipeline has at least one job.

A job is a series of tasks that run sequentially on the same target. It can be a Windows server, a Linux server, a container, or a deployment group.

A release job is executed by a build/release agent. This agent can only run one job at the same time.

You specify a series of tasks you want to run on the same agent during your job design.

When the build or release pipeline is triggered at runtime, each job is dispatched to its target as one or more.

A scenario that speaks to the imagination, where Jobs plays an essential role, is the following.

Assume that you built an application with a backend in .NET, a front end in Angular, and a native IOS mobile App. It might be developed in three different source control repositories triggering three other builds and delivering three other artifacts.

The release pipeline brings the artifacts together and wants to deploy the backend, frontend, and Mobile App all together as part of one release.

The deployment needs to take place on different agents.

If an IOS app needs to be built and distributed from a Mac, the angular app is hosted on Linux, so best deployed from a Linux machine.

The backend might be deployed from a Windows machine.

Because you want all three deployments to be part of one pipeline, you can define multiple Release Jobs targeting the different agents, servers, or deployment groups.

By default, jobs run on the host machine where the agent is installed.

It's convenient and typically well suited for projects just beginning to adopt continuous integration (CI).

Over time, you may want more control over the stage where your tasks run.

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

For more information, see [Jobs in Azure Pipelines](#).

## Configure Pipelines as Code with YAML

Completed 100 XP

- 60 minutes

**Estimated time:** 60 minutes.

**Lab files:** none.

## Scenario

Many teams prefer to define their build and release pipelines using YAML. This allows them to access the same pipeline features as those using the visual designer but with a markup file that can be managed like any other source file. YAML build definitions can be added to a project by simply adding the corresponding files to the repository's root. Azure DevOps also provides default templates for popular project types and a YAML designer to simplify the process of defining build and release tasks.

## Objectives

After completing this lab, you'll be able to:

- Configure CI/CD pipelines as code with YAML in Azure DevOps.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft account or a Microsoft Entra account with the Owner role in the Azure subscription and the Global Administrator role in the Microsoft Entra tenant associated with the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](#) and [View and assign administrator roles in Microsoft Entra ID](#).

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Configure CI/CD Pipelines as Code with YAML in Azure DevOps.
- Exercise 2: Configure Environment settings for CI/CD Pipelines as Code with YAML in Azure DevOps.
- Exercise 3: Remove the Azure lab resources.

# Introduction to deployment patterns

## Explore microservices architecture

Completed 100 XP

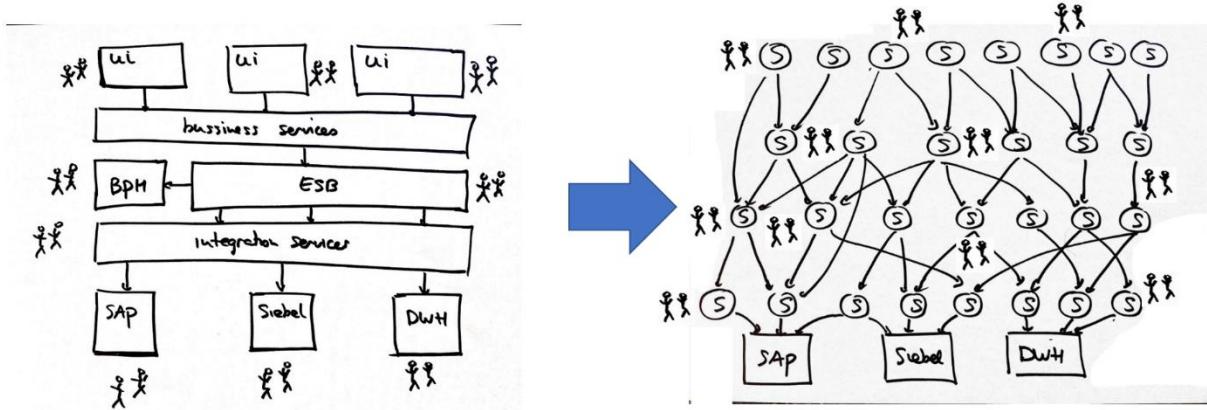
- 2 minutes

Today, you'll frequently hear the term microservices. A microservice is an autonomous, independently deployable, and scalable software component.

They're small, focused on doing one thing well, and can run autonomously. If one microservice changes, it shouldn't impact any other microservices within your landscape.

By choosing a microservices architecture, you'll create a landscape of services that can be developed, tested, and deployed separately. It implies other risks and complexity.

It would be best if you created it to keep track of interfaces and how they interact. And you need to maintain multiple application lifecycles instead of one.



In a traditional application, we can often see a multi-layer architecture.

One layer with the UI, a layer with the business logic and services, and a layer with the data services.

Sometimes there are dedicated teams for the UI and the backend. When something needs to change, it needs to change in all the layers.

When moving towards a microservices architecture, all these layers are part of the same microservice.

Only the microservice contains one specific function.

The interaction between the microservices is done asynchronously.

They don't call each other directly but use asynchronous mechanisms like queues or events.

Each microservice has its lifecycle and Continuous Delivery pipeline. If you built them correctly, you could deploy new microservice versions without impacting other system parts.

Microservice architecture is undoubtedly not a prerequisite for Continuous Delivery, but smaller software components help implement a fully automated pipeline.

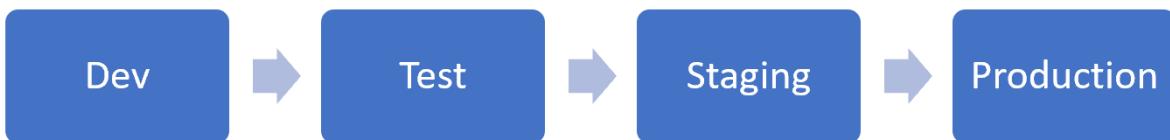
## Examine classical deployment patterns

Completed 100 XP

- 1 minute

When we have our prerequisites to deliver our software continuously, we need to start thinking about a deployment pattern.

Traditionally a deployment pattern was straightforward.



The software was built, and when all features had been implemented, the software was deployed to an environment where a group of people could start using it.

The traditional or classical deployment pattern was moving your software to a development stage, a testing stage, maybe an acceptance or staging stage, and finally a production stage.

The software moved as one piece through the stages.

The production release was, in most cases, a Big Bang release, where users were confronted with many changes at the same time.

Despite the different stages to test and validate, this approach still involves many risks.

By running all your tests and validation on non-production environments, it's hard to predict what happens when your production users start using it.

You can run load tests and availability tests, but in the end, there's no place like production.

## Understand modern deployment patterns

Completed 100 XP

- 1 minute

End-users always use your application differently. Unexpected events will happen in a data center, multiple events from multiple users will cooccur, triggering some code that hasn't been tested in that way.

To overcome, we need to embrace that some features can only be tested in production.

Testing in production sounds a bit scary, but that shouldn't be the case.

When we talked about separating our functional and technical releases, we already saw that it's possible to deploy features without exposing them to all users.

When we take this concept of feature toggling and use it with our deployment patterns, we can test our software in production.

For example:

- Blue-green deployments.
- Canary releases.
- Dark launching.
- A/B testing.
- Progressive exposure or ring-based deployment.
- Feature toggles.

## Take a critical look at your architecture

Are your architecture and the current state of your software ready for Continuous Delivery?

Topics you might want to consider are:

- Is your software built as one giant monolith, or is it divided into multiple components?
- Can you deliver parts of your application separately?
- Can you guarantee the quality of your software when deploying multiple times a week?
- How do you test your software?
- Do you run one or multiple versions of your software?
- Can you run multiple versions of your software side by side?
- What do you need to improve to implement Continuous Delivery?

# Manage application configuration data

## Rethink application configuration data

Completed 100 XP

- 4 minutes

### Configuration information in files

Most application runtime environments include configuration information that is held in files deployed with the application. In some cases, it's possible to edit these files to change the application behavior after deploying.

However, changes to the configuration require the application to be redeployed, often resulting in unacceptable downtime and other administrative overhead.

Local configuration files also limit the configuration to a single application, but sometimes it would be helpful to share configuration settings across multiple applications.

Examples include database connection strings, UI theme information, or the URLs of queues and storage used by a related set of applications.

It's challenging to manage changes to local configurations across multiple running instances of the application, especially in a cloud-hosted scenario.

It can result in instances using different configuration settings while the update is being deployed. Also, updates to applications and components might require changes to configuration schemas.

Many configuration systems don't support different versions of configuration information.

### Example

It's 2:00 AM. Adam is done making all changes to his super fantastic code piece.

The tests are all running fine. They hit commit -> push -> all commits pushed successfully to git. Happily, Adam drives back home. 10 mins later, they get a call from the SecurityOps engineer, "Adam, did you push the Secret Key to our public repo?"

YIKES! That blah.config file, Adam thinks. How could I've forgotten to include that in .gitignore? The nightmare has already begun.

We can surely blame Adam for sinning, checking in sensitive secrets, and not following the recommended practices of managing configuration files.

Still, the bigger question is that if the underlying toolchain had abstracted out the configuration management from the developer, this fiasco would have never happened!

## History

The virus was injected a long time ago. Since the early days of .NET, the app.config and web.config files have the notion that developers can make their code flexible by moving typical configuration into these files.

When used effectively, these files are proven to be worthy of dynamic configuration changes. However, much time, we see the misuse of what goes into these files.

A common culprit is how samples and documentation have been written. Most samples on the web would usually use these config files to store key elements such as ConnectionStrings and even passwords.

The values might be obfuscated but what we are telling developers is that "hey, It's a great place to push your secrets!".

So, in a world where we're preaching using configuration files, we can't blame the developer for not managing its governance.

We aren't challenging the use of Configuration here. It's an absolute need for an exemplary implementation. Instead, we should debate using multiple JSON, XML, and YAML files to maintain configuration settings.

Configs are great for ensuring the flexibility of the application, config files. However, they aren't straightforward, especially across environments.

## A ray of hope: The DevOps movement

In recent years, we've seen a shift around following some excellent practices around effective DevOps and some great tools (Chef, Puppet) for managing Configuration for different languages.

While these have helped inject values during CI/CD pipeline and greatly simplified configuration management, the blah.config concept hasn't moved away.

Frameworks like ASP.NET Core support the notion of appSettings.json across environments.

The framework has made it practical to use across environments through interfaces like IHostingEnvironment and IConfiguration, but we can do better.

## Explore separation of concerns

Completed 100 XP

- 2 minutes

One of the key reasons we would want to move the configuration away from source control is to outline responsibilities.

Let's define some roles to elaborate on them. None of those are new concepts but rather a high-level summary:

- **Configuration custodian:** Responsible for generating and maintaining the life cycle of configuration values. These include CRUD on keys, ensuring the security of secrets, regeneration of keys and tokens, defining configuration settings such as Log levels for each environment. This role can be owned by operation engineers and security engineering while injecting configuration files through proper DevOps processes and CI/CD implementation. They do not define the actual configuration but are custodians of their management.
- **Configuration consumer:** Responsible for defining the schema (loose term) for the configuration that needs to be in place and then consuming the configuration values in the application or library code. It's the Dev. And Test teams shouldn't be concerned about the value of keys but rather what the key's capability is. For example, a developer may need a different

ConnectionString in the application but not know the actual value across different environments.

- **Configuration store:** The underlying store used to store the configuration, while it can be a simple file, but in a distributed application, it needs to be a reliable store that can work across environments. The store is responsible for persisting values that modify the application's behavior per environment but aren't sensitive and don't require any encryption or HSM modules.
- **Secret store:** While you can store configuration and secrets together, it violates our separation of concern principle, so the recommendation is to use a different store for persisting secrets. It allows a secure channel for sensitive configuration data such as ConnectionStrings, enables the operations team to have Credentials, Certificate, Token in one repository, and minimizes the security risk if the Configuration Store gets compromised.

## Understand external configuration store patterns

Completed 100 XP

- 3 minutes

These patterns store the configuration information in an external location and provide an interface that can be used to quickly and efficiently read and update configuration settings.

The type of external store depends on the hosting and runtime environment of the application.

A cloud-hosted scenario is typically a cloud-based storage service but could be a hosted database or other systems.

The backing store you choose for configuration information should have an interface that provides consistent and easy-to-use access.

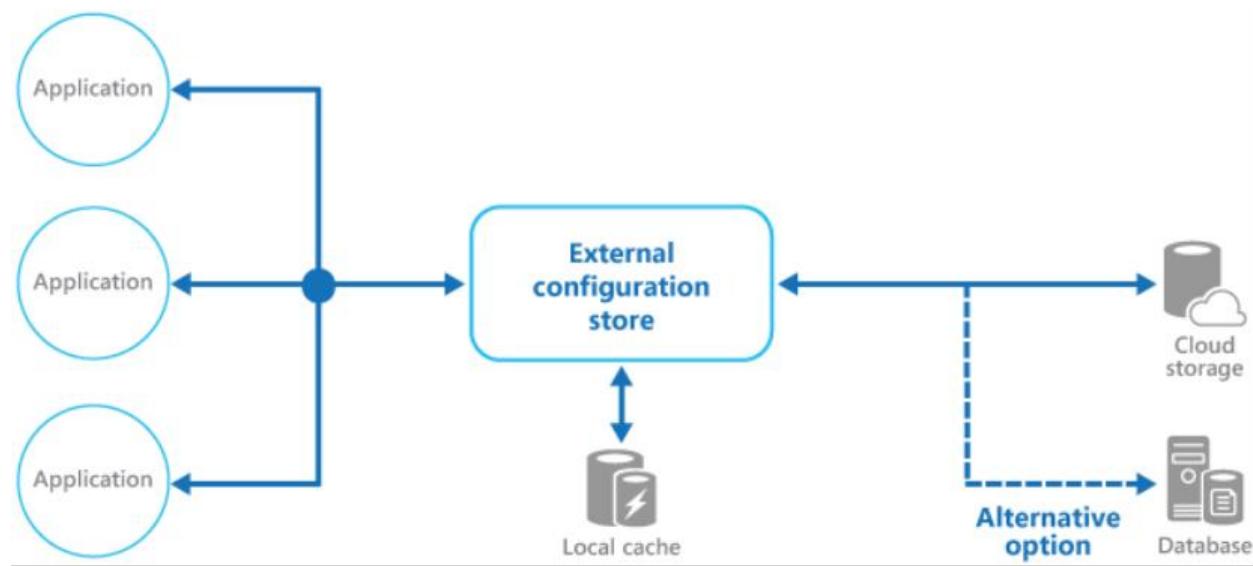
It should expose the information in a correctly typed and structured format.

The implementation might also need to authorize users' access to protect configuration data and be flexible enough to allow storage of multiple configuration versions (such as development, staging, or production, including many release versions of each one).

Many built-in configuration systems read the data when the application starts up and cache the data in memory to provide fast access and minimize the impact on application performance.

Depending on the type of backing store used and its latency, it might be helpful to implement a caching mechanism within the external configuration store.

For more information, see the Caching Guidance. The figure illustrates an overview of the External Configuration Store pattern with optional local cache.



This pattern is helpful for:

- Configuration settings are shared between multiple applications and application instances, or where a standard configuration must be enforced across various applications and application instances.
- A standard configuration system doesn't support all the required configuration settings, such as storing images or complex data types.
- As a complementary store for some application settings, they allow applications to override some or all the centrally stored settings.
- To simplify the administration of multiple applications and optionally monitor configuration settings by logging some or all types of access to the configuration store.

# Introduction to Azure App Configuration

100 XP

- 3 minutes

Azure App Configuration is a service for central management of application settings and feature flags.

Modern programs include distributed components, each that needs its settings.

It's prevalent with microservice-based applications and with serverless applications.

Distributed configuration settings can lead to hard-to-troubleshoot deployment errors.

Azure App Configuration service stores all the settings for your application and secures their access in one place.

Azure App Configuration service provides the following features:

- A fully managed service that can be set up in minutes.
- Flexible key representations and mappings.
- Tagging with labels.
- A point-in-time replay of settings.
- Dedicated UI for feature flag management.
- Comparison of two sets of configurations on custom-defined dimensions.
- Enhanced security through Azure managed identities.
- Complete data encryptions, at rest or in transit.
- Native integration with popular frameworks.

App Configuration complements Azure Key Vault, which is used to store application secrets. App Configuration makes it easier to implement the following scenarios:

- Centralize management and distribution of hierarchical configuration data for different environments and geographies.
- Dynamically change application settings without the need to redeploy or restart an application.
- Control feature availability in real time.

## Use App Configuration

The easiest way to add an App Configuration store to your application is through one of Microsoft's client libraries.

Based on the programming language and framework, the following best methods are available to you.

Expand table

#### Programming language and framework

.NET Core and ASP.NET Core

.NET Framework and ASP.NET

Java Spring

Others

#### How to connect

App Configuration provider for .NET Core

App Configuration builder for .NET

App Configuration client for Spring Cloud

App Configuration REST API

## Examine Key-value pairs

100 XP

- 4 minutes

Azure App Configuration stores configuration data as key-value pairs.

## Keys

Keys serve as the name for key-value pairs and are used to store and retrieve corresponding values.

It's common to organize keys into a hierarchical namespace by using a character delimiter, such as / or :. Use a convention that's best suited for your application.

App Configuration treats keys as a whole. It doesn't parse keys to figure out how their names are structured or enforce any rule on them.

Keys stored in App Configuration are case-sensitive, Unicode-based strings.

The keys *app1* and *App1* are distinct in an App Configuration store.

When you use configuration settings within an application, keep it in mind because some frameworks handle configuration keys case-insensitively.

You can use any Unicode character in key names entered into App Configuration except for \*, ,, and \.

These characters are reserved. If you need to include a reserved character, you must escape it by using \{Reserved Character}.

There's a combined size limit of 10,000 characters on a key-value pair.

This limit includes all characters in the key, its value, and all associated optional attributes.

Within this limit, you can have many hierarchical levels for keys.

## Design key namespaces

There are two general approaches to naming keys used for configuration data: flat or hierarchical.

These methods are similar from an application usage standpoint, but hierarchical naming offers several advantages:

- Easier to read. Instead of one long sequence of characters, delimiters in a hierarchical key name function as spaces in a sentence.
- Easier to manage. A key name hierarchy represents logical groups of configuration data.
- Easier to use. It's simpler to write a query that pattern-matches keys in a hierarchical structure and retrieves only a portion of configuration data.

Below are some examples of how you can structure your key names into a hierarchy:

- Based on component services

Copy

```
AppName:Service1:ApiEndpoint  
AppName:Service2:ApiEndpoint
```

- Based on deployment regions

Copy

```
AppName:Region1:DbEndpoint  
AppName:Region2:DbEndpoint
```

## Label keys

Key values in App Configuration can optionally have a label attribute.

Labels are used to differentiate key values with the same key.

A key *app1* with labels *A* and *B* forms two separate keys in an App Configuration store.

By default, the label for a key value is empty or null.

Label provides a convenient way to create variants of a key. A common use of labels is to specify multiple environments for the same key:

Copy

```
Key = AppName:DbEndpoint & Label = Test  
Key = AppName:DbEndpoint & Label = Staging  
Key = AppName:DbEndpoint & Label = Production
```

## Version key values

App Configuration doesn't version key values automatically as they're modified.

Use labels as a way to create multiple versions of a key value.

For example, you can input an application version number or a Git commit ID in labels to identify key values associated with a particular software build.

## Query key values

Each key value is uniquely identified by its key plus a label that can be `null`. You query an App Configuration store for key values by specifying a pattern.

The App Configuration store returns all key values that match the pattern and their corresponding values and attributes.

# Values

Values assigned to keys are also Unicode strings. You can use all Unicode characters for values.

There's an optional user-defined content type associated with each value.

Use this attribute to store information, for example, an encoding scheme, about a value that helps your application process it properly.

Configuration data stored in an App Configuration store, which includes all keys and values, is encrypted at rest and in transit.

App Configuration isn't a replacement solution for Azure Key Vault. Don't store application secrets in it.

# Examine App configuration feature management

100 XP

- 4 minutes

Feature management is a modern software development practice that decouples feature release from code deployment and enables quick changes to feature availability on demand.

Feature Flags are discussed in another module, but at this point, it's worth noting that Azure App Configuration Service can be used to store and manage feature flags. (It's also known as feature toggles, feature switches, and other names).

## Basic concepts

Here are several new terms related to feature management:

- **Feature flag:** A feature flag is a variable with a binary state of *on* or *off*. The feature flag also has an associated code block. The state of the feature flag triggers whether the code block runs or not.
- **Feature manager:** A feature manager is an application package that handles the lifecycle of all the feature flags in an application. The feature manager typically provides more functionality, such as caching feature flags and updating their states.

- **Filter:** A filter is a rule for evaluating the state of a feature flag. A user group, a device or browser type, a geographic location, and a time window are all examples of what a filter can represent.

Effective implementation of feature management consists of at least two components working in concert:

- An application that makes use of feature flags.
- A separate repository that stores the feature flags and their current states.

How these components interact is illustrated in the following examples.

## Feature flag usage in code

The basic pattern for implementing feature flags in an application is simple. You can think of a feature flag as a Boolean state variable used with an `if` conditional statement in your code:

```
C#Copy
if (featureFlag) {
    // Run the following code.
}
```

In this case, if `featureFlag` is set to `True`, the enclosed code block is executed; otherwise, it's skipped. You can set the value of `featureFlag` statically, as in the following code example:

```
C#Copy
bool featureFlag = true;
```

You can also evaluate the flag's state based on certain rules:

```
C#Copy
bool featureFlag = isBetaUser();
```

A slightly more complicated feature flag pattern includes an `else` statement as well:

```
C#Copy
if (featureFlag) {
    // This following code will run if the featureFlag value is true.
```

```
} else {
    // This following code will run if the featureFlag value is false.
}
```

## Feature flag declaration

Each feature flag has two parts: a name and a list of one or more filters used to evaluate if a feature's state is *on* (that is when its value is `True`).

A filter defines a use case for when a feature should be turned on.

When a feature flag has multiple filters, the filter list is traversed until one of the filters determines the feature should be enabled.

At that point, the feature flag is *on*, and any remaining filter results are skipped. If no filter indicates the feature should be enabled, the feature flag is *off*.

The feature manager supports `appsettings.json` as a configuration source for feature flags.

The following example shows how to set up feature flags in a JSON file:

JSONCopy

```
"FeatureManagement": {
    "FeatureA": true, // Feature flag set to on
    "FeatureB": false, // Feature flag set to off
    "FeatureC": {
        "EnabledFor": [
            {
                "Name": "Percentage",
                "Parameters": {
                    "Value": 50
                }
            }
        ]
    }
}
```

## Feature flag repository

To use feature flags effectively, you need to externalize all the feature flags used in an application.

This approach allows you to change feature flag states without modifying and redeploying the application itself.

Azure App Configuration is designed to be a centralized repository for feature flags.

You can use it to define different kinds of feature flags and manipulate their states quickly and confidently.

You can then use the App Configuration libraries for various programming language frameworks to easily access these feature flags from your application.

## Integrate Azure Key Vault with Azure Pipelines

Completed 100 XP

- 1 minute

Applications contain many secrets, such as:

- Connection strings.
- Passwords.
- Certificates.
- Tokens, which, if leaked to unauthorized users, can lead to a severe security breach.

It can result in severe damage to the organization's reputation and compliance issues with different governing bodies.

Azure Key Vault allows you to manage your organization's secrets and certificates in a centralized repository.

The secrets and keys are further protected by Hardware Security Modules (HSMs).

It also provides versioning of secrets, full traceability, and efficient permission management with access policies.

For more information on Azure Key Vault, visit [What is Azure Key Vault](#).

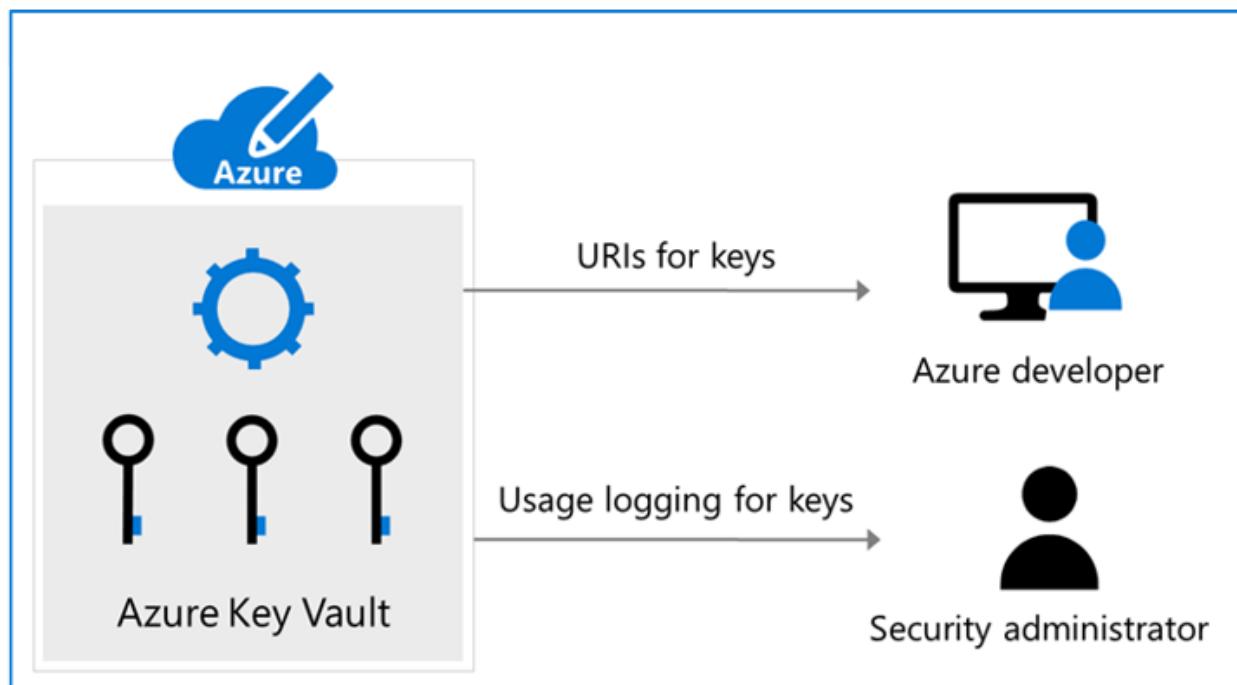
# Manage secrets, tokens and certificates

100 XP

- 4 minutes

Azure Key Vault helps solve the following problems:

- **Secrets management** - Azure Key Vault can be used to store securely and tightly control access to tokens, passwords, certificates, API keys, and other secrets.
- **Key management** - Azure Key Vault can also be used as a key management solution. Azure Key Vault makes it easy to create and control the encryption keys used to encrypt your data.
- **Certificate management** - Azure Key Vault is also a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with Azure. And your internal connected resources.
- **Store secrets backed by hardware security modules** - The secrets and keys can be protected by software or FIPS 140-2 Level 2 validates HSMs.



## Why use Azure Key Vault?

## Centralize application secrets

Centralizing the storage of application secrets in Azure Key Vault allows you to control their distribution.

Key Vault dramatically reduces the chances that secrets may be accidentally leaked.

When using Key Vault, application developers no longer need to store security information in their applications. It eliminates the need to make this information part of the code.

For example, an application may need to connect to a database. Instead of storing the connection string in the app codes, store it securely in Key Vault.

Your applications can securely access the information they need by using URLs that allow them to retrieve specific versions of a secret after the application's key or secret is stored in Azure Key Vault.

It happens without having to write custom code to protect any of the secret information.

## Securely store secrets and keys

Secrets and keys are safeguarded by Azure, using industry-standard algorithms, key lengths, and hardware security modules (HSMs).

The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated.

Access to a key vault requires proper authentication and authorization before a caller (user or application) can get access.

Authentication establishes the identity of the caller, while authorization determines the operations that they can do.

Authentication is done via Microsoft Entra ID. Authorization may be done via role-based access control (RBAC) or Key Vault access policy.

RBAC is used when dealing with the management of the vaults, and a key vault access policy is used when attempting to access data stored in a vault.

Azure Key Vaults may be either software- or hardware-HSM protected.

You can import or generate keys in hardware security modules (HSMs) that never leave the HSM boundary when you require added assurance.

Microsoft uses Thales hardware security modules. You can use Thales tools to move a key from your HSM to Azure Key Vault.

Finally, Azure Key Vault is designed so that Microsoft doesn't see or extract your data.

## Monitor access and use

Once you've created a couple of Key Vaults, you'll want to monitor how and when your keys and secrets are accessed.

You can do it by enabling logging for Key Vault. You can configure Azure Key Vault to:

- Archive to a storage account.
- Stream to an Event Hubs.
- Send the logs to Log Analytics.

You have control over your logs, and you may secure them by restricting access, and you may also delete logs that you no longer need.

## Simplified administration of application secrets

When storing valuable data, you must take several steps. Security information must be secured. It must follow a lifecycle. It must be highly available.

Azure Key Vault simplifies it by:

- Removing the need for in-house knowledge of Hardware Security Modules.
- Scaling up on short notice to meet your organization's usage spikes.
- Replicating the contents of your Key Vault within a region and to a secondary region. It ensures high availability and takes away the need for any action from the administrator to trigger the failover.
- Providing standard Azure administration options via the portal, Azure CLI and PowerShell.

- Automating specific tasks on certificates that you purchase from Public CAs, such as enrollment and renewal.

Also, Azure Key Vaults allow you to segregate application secrets.

Applications may access only the vault they can access, and they can only do specific operations.

You can create an Azure Key Vault per application and restrict the secrets stored in a Key Vault to a particular application and team of developers.

## Integrate with other Azure services

As a secure store in Azure, Key Vault has been used to simplify scenarios like Azure Disk Encryption, the always encrypted functionality in SQL Server and Azure SQL Database, Azure web apps.

Key Vault itself can integrate with storage accounts, Event Hubs, and log analytics.

## Examine DevOps inner and outer loop

Completed 100 XP

- 2 minutes

While you can store configuration and secrets together, it violates our separation of concern principle, so the recommendation is to use a different store for persisting secrets.

It allows a secure channel for sensitive configuration data, such as ConnectionStrings.

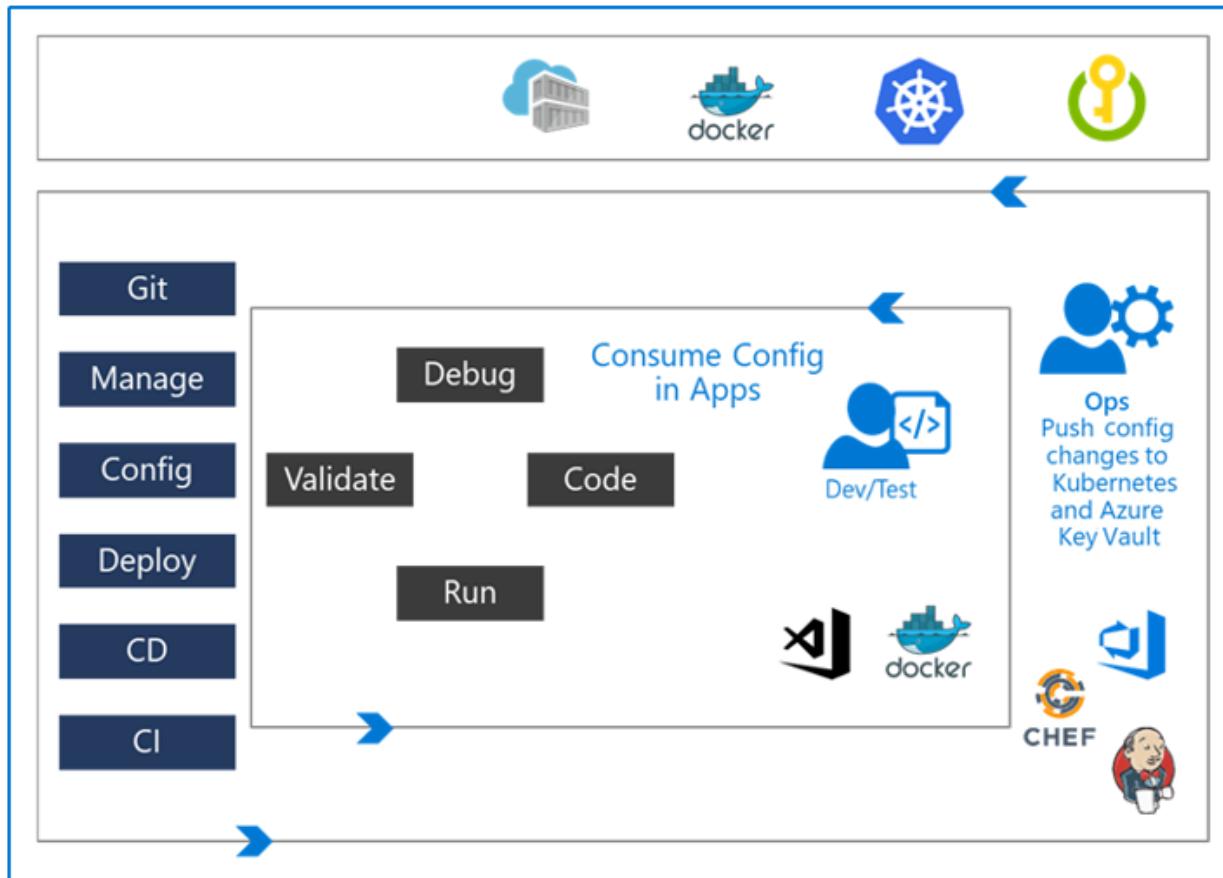
It enables the operations team to have credentials, certificates, and tokens in one repository and minimizes the security risk if the Configuration Store gets compromised.

The following diagram shows how these roles play together in a DevOps inner and outer loop.

The inner loop is focused on the developer teams iterating over their solution development; they consume the configuration published by the Outer Loop.

The Ops Engineer governs the Configuration management.

They push changes into Azure KeyVault and Kubernetes that are further isolated per environment.



## Integrate Azure Key Vault with Azure DevOps

Completed 100 XP

- 40 minutes

**Estimated time:** 40 minutes.

**Lab files:** none.

## Scenario

Azure Key Vault provides secure storage and management of sensitive data, such as keys, passwords, and certificates. Azure Key Vault includes support for hardware security modules and a range of encryption algorithms and key lengths. Using Azure Key Vault can minimize the possibility of disclosing sensitive data through source code, a common mistake developers make. Access to Azure Key Vault requires proper authentication and authorization, supporting fine-grained permissions to its content.

In this lab, you'll see how you can integrate Azure Key Vault with an Azure Pipeline by using the following steps:

- Create an Azure Key vault to store an ACR password as a secret.
- Create an Azure Service Principal to access Azure Key Vault's secrets.
- Configure permissions to allow the Service Principal to read the secret.
- Configure the pipeline to retrieve the password from the Azure Key Vault and pass it on to subsequent tasks.

## Objectives

After completing this lab, you'll be able to:

- Create a Microsoft Entra service principal.
- Create an Azure Key Vault.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft or Microsoft Entra account with the Owner role in the Azure subscription and the Global Administrator role in the Microsoft Entra tenant associated with the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](#) and [View and assign administrator roles in Microsoft Entra ID](#).

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Setup CI Pipeline to build eShopOnWeb container.
- Exercise 2: Remove the Azure lab resources.

# Integrate with identity management systems

## Integrate GitHub with single sign-on (SSO)

Completed 100 XP

- 2 minutes

To use SSO, you need to connect your identity provider to GitHub at the organization level.

GitHub offers both **SAML** and **SCIM** support.

| Provider                                    | Available Support |
|---|-------------------|
| Active Directory Federation Services (ADFS) | SAML              |
| Microsoft Entra ID                          | SAML and SCIM     |
| Okta  | SAML and SCIM     |
| OneLogin                                    | SAML and SCIM     |
| PingOne                                     | SAML              |
| Shibboleth                                  | SAML              |

For more information, see:

- [Enforcing SAML single sign-on for your organization](#).

## Explore service principals

Completed 100 XP

- 2 minutes

Microsoft Entra ID offers different kinds of mechanisms for authentication. In DevOps Projects, though, one of the most important is the use of Service Principals.

## Microsoft Entra applications

Applications are registered with a Microsoft Entra tenant within Microsoft Entra ID. Registering an application creates an identity configuration. You also determine who can use it:

- Accounts in the same organizational directory.
- Accounts in any organizational directory.
- Accounts in any organizational directory and Microsoft Accounts (personal).
- Microsoft Accounts (Personal accounts only).

The user-facing display name for this application (this can be changed later).

### Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (SQL Down Under Pty Ltd only - Single tenant)  
 Accounts in any organizational directory (Any Azure AD directory - Multitenant)  
 Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)  
 Personal Microsoft accounts only

[Help me choose...](#)

### Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web  e.g. <https://myapp.com/auth>

All applications    Owned applications

Start typing a name or Application ID to filter these results

| Display name    | Application (client) ID | Created on | Certificates & secrets |
|-----------------|-------------------------|------------|------------------------|
| AC    ACMOrders | 02744c0i                | 5caf       | 11/6/2019              |

## Client secret

Once the application is created, you then should create at least one client secret for the application.

#### Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

| New client secret   |            |          |
|---------------------|------------|----------|
| Description         | Expires    | Value    |
| ACM Orders Download | 12/31/2299 | mXL***** |

## Grant permissions

The application identity can then be granted permissions within services and resources that trust Microsoft Entra ID.

## Service principal

To access resources, an entity must be represented by a security principal. To connect, the entity must know:

- TenantID.
- ApplicationID.
- Client Secret.

For more information on Service Principals, see [App Objects and Service Principals](#).

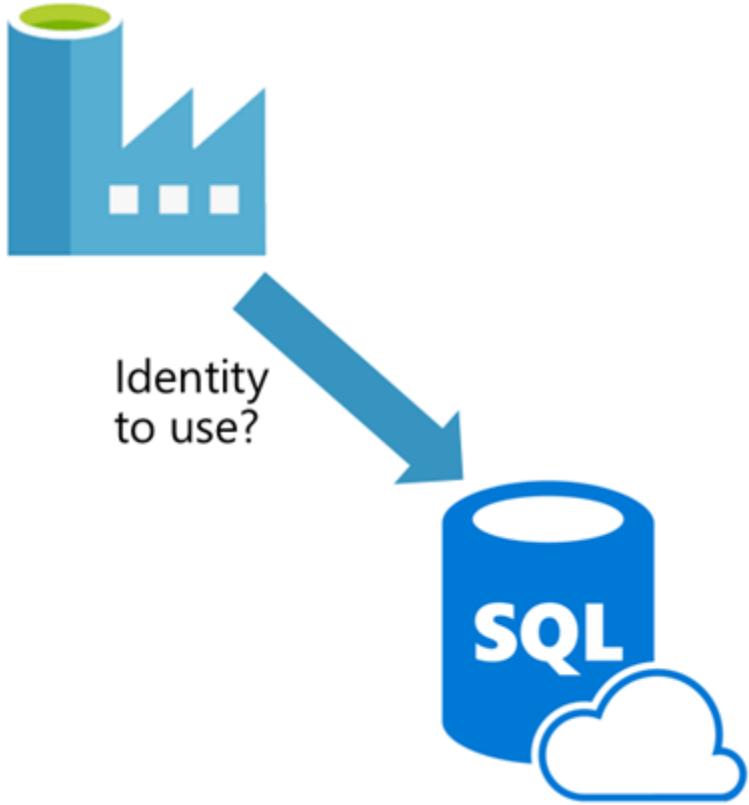
## Explore Managed Identity

Completed 100 XP

- 2 minutes

Another authentication mechanism offered by Microsoft Entra ID is Managed identities.

Imagine that you need to connect from an Azure Data Factory (ADF) to an Azure SQL Database. What identity should ADF present to the database?



The traditional answer would have been to use SQL Authentication with a username and password. It leaves yet another credential that needs to be managed on an ongoing basis.

## Identity of the service

Many Azure services expose their own identity. It isn't an identity that you need to manage. For example, you don't need to worry about password policies and so on.

You can assign permissions to that identity, as with any other Microsoft Entra identity.

In the ADF example, you can add the ADF MSI as an Azure SQL Database user and add it to roles within the database.

## Managed identity types

There are two types of managed identities:

- **System-assigned** - It's the types of identities described above. Many, but not all, services expose these identities.
- **User-assigned** - you can create a managed identity as an Azure resource. It can then be assigned to one or more instances of a service.

For more information, see: [What are managed identities for Azure resources?](#)

## Implement A/B testing and progressive exposure deployment

### What is A/B testing?

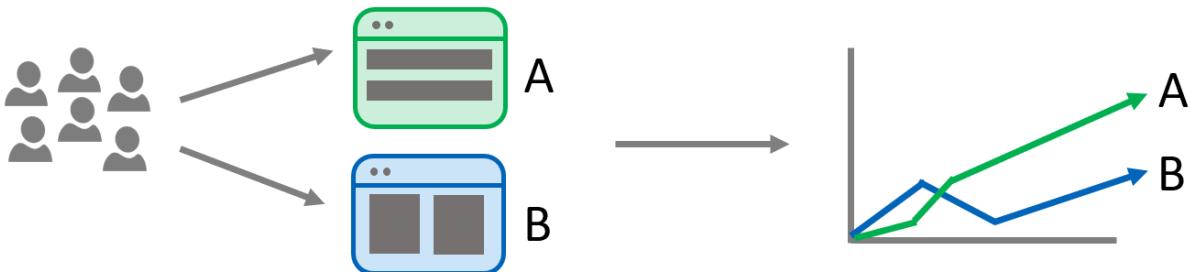
Completed 100 XP

- 1 minute

A/B testing (also known as split testing or bucket testing) compares two versions of a web page or app against each other to determine which one does better.

A/B testing is mainly an experiment where two or more page variants are shown to users at random.

Also, statistical analysis is used to determine which variation works better for a given conversion goal.



A/B testing isn't part of continuous delivery or a pre-requisite for continuous delivery. It's more the other way around.

Continuous delivery allows you to deliver MVPs to a production environment and your end-users quickly.

Common aims are to experiment with new features, often to see if they improve conversion rates.

Experiments are continuous, and the impact of change is measured.

A/B testing is out of scope for this course.

But because it's a powerful concept that is enabled by implementing continuous delivery, it's mentioned here to dive into further.

## Explore CI-CD with deployment rings

Completed 100 XP

- 2 minutes

Progressive exposure deployment, also called ring-based deployment, was first discussed in Jez Humble's Continuous Delivery book.

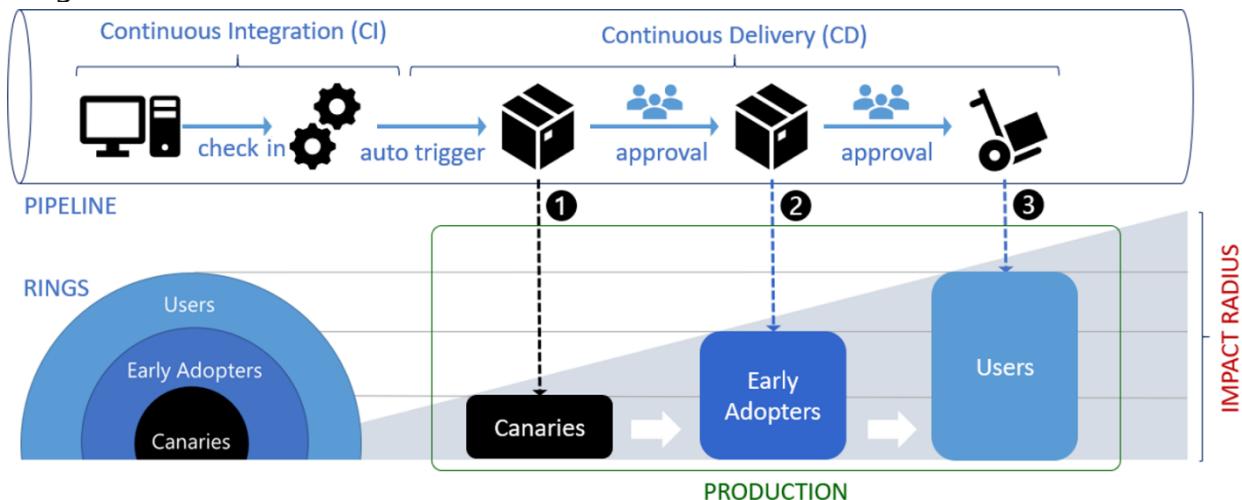
They support the production-first DevOps mindset and limit the impact on end users while gradually deploying and validating changes in production.

**Impact (also called blast radius)** is evaluated through observation, testing, analysis of telemetry, and user feedback.

In DevOps, rings are typically modeled as stages.

Rings are, in essence, an extension of the canary stage. The canary release releases to a stage to measure impact. Adding another ring is essentially the same

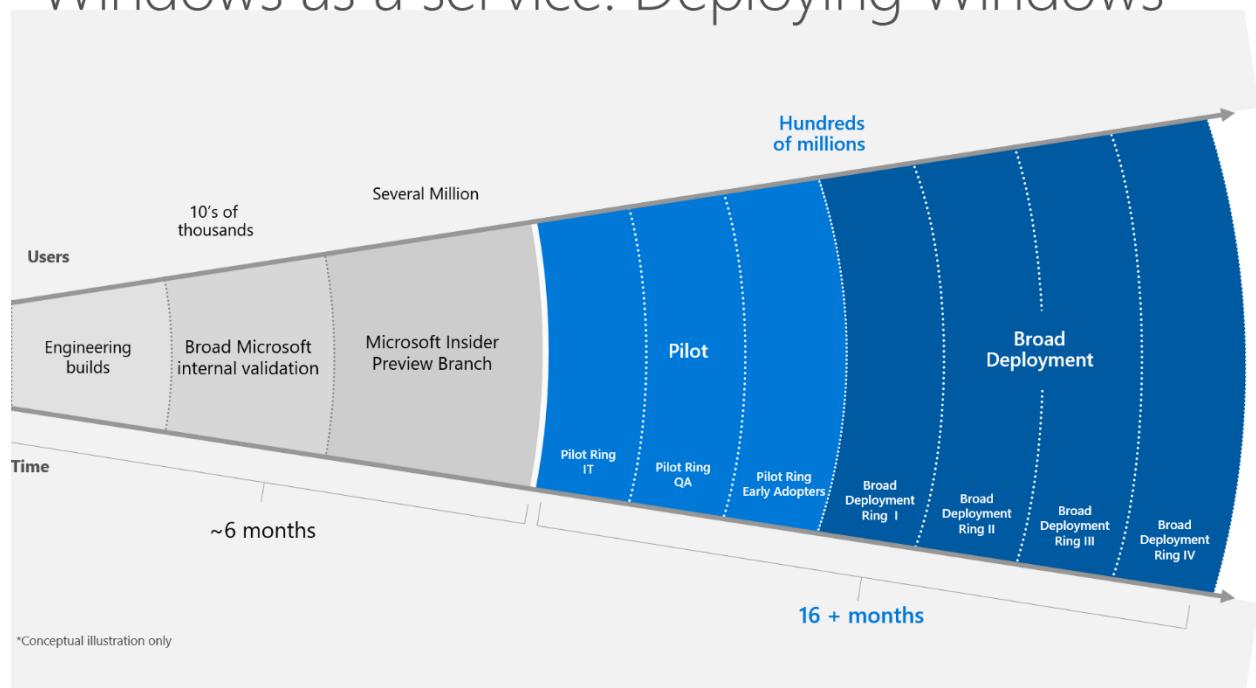
thing.



With a ring-based deployment, you first deploy your changes to risk-tolerant customers and progressively roll out to a more extensive set of customers.

The Microsoft Windows team, for example, uses these rings.

## Windows as a service: Deploying Windows



When you have identified multiple groups of users and see value in investing in a ring-based deployment, you need to define your setup.

Some organizations that use canary releasing have multiple deployment slots set up as rings.

The first release of the feature to ring 0 targets a well-known set of users, mostly their internal organization.

After things have been proven stable in ring 0, they propagate the release to the next ring. It's with a limited set of users outside their organization.

And finally, the feature is released to everyone. It is often done by flipping the switch on the feature toggles in the software.

As in the other deployment patterns, monitoring and health checks are essential.

By using post-deployment release gates that check a ring for health, you can define an automatic propagation to the next ring after everything is stable.

When a ring isn't healthy, you can halt the deployment to the following rings to reduce the impact.

For more information, see also [Explore how to progressively expose your Azure DevOps extension releases in production to validate before impacting all users](#).

## Exercise - Ring-based deployment

Completed 100 XP

- 5 minutes

In this exercise, you'll investigate ring-based deployment.

### Steps

Let's look at how a release pipeline can stage features using ring-based deployments.

When I have a new feature, I might want to release it to a few users first, just in case something goes wrong.

I could do it in authenticated systems by having those users as members of a security group and letting members of that group use the new features.

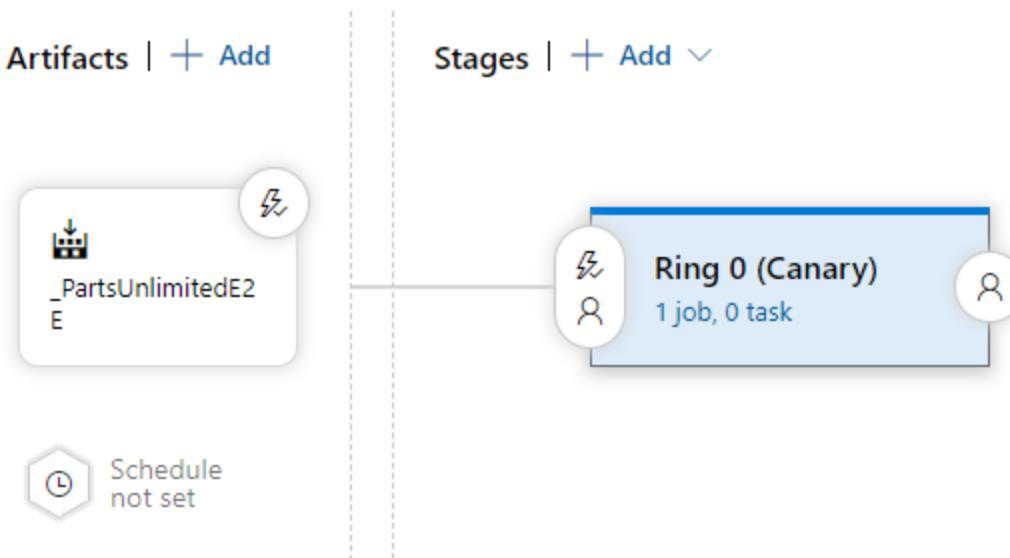
However, on a public website, I might not have logged-in users. Instead, I might want to direct a small percentage of the traffic to use the new features.

Let's see how that's configured.

We'll create a new release pipeline that isn't triggered by code changes but manually when we slowly release a new feature.

We start by assuming that a new feature has already been deployed to the Green site (the staging slot).

1. In the main menu for the **PU Hosted** project, click **Pipelines**, then click **Release**, click **+New**, then click **New release pipeline**.
2. When prompted to select a template, click **Empty job** from the top of the pane.
3. Click on the **Stage 1** stage and rename it to **Ring 0 (Canary)**.



4. Hover over the **New release pipeline** name at the top of the page, and when a pencil appears, click it, and change the pipeline name to **Ring-based Deployment**.

All pipelines > **Ring-based Deployment**

[Pipeline](#) [Tasks](#) [Variables](#) [Retention](#) [Options](#) [History](#)

- Select the **Ring 0 (Canary)** stage from the Tasks drop-down list. Click the + to add a new task, and from the list of tasks, hover over **Azure CLI** when the **Add** button appears, click it, then click to select the **Azure CLI** task in the task list for the stage.

The screenshot shows the configuration for an Azure CLI task. At the top right are 'View YAML' and 'Remove' buttons. Below is a dropdown for 'Task version' set to '1.\*'. The 'Display name' field contains 'Azure CLI'. Under 'Azure subscription', there is a dropdown with a red border and an error message: 'This setting is required.' The 'Script Location' dropdown is set to 'Inline script'. The 'Script path' dropdown is empty. The 'Script Path' dropdown is also empty with an error message: 'This setting is required.' A '...' button is at the bottom right of the path dropdown.

- In the **Azure CLI** settings pane, select your **Azure subscription**, set **Script Location** to **Inline script**, set the **Inline Script** to the following, then click **Save** and **OK**.

```
az webapp traffic-routing set --resource-group $(ResourceGroupName) --name $(WebsiteName) --distribution staging=10
```

This distribution will cause 10% of the web traffic to be sent to the new feature Site (currently the staging slot).

- From the menu above the task list, click **Variables**. Create two new variables as shown. (Make sure to use your correct website name).

| Name              | Value          | Scope   |
|-------------------|----------------|---------|
| ResourceGroupName | ASPDOTNET      | Release |
| WebsiteName       | pule2eb3100890 | Release |

- From the menu above the variables, click **Pipeline** to return to editing the pipeline. Hover over the **Ring 0 (Canary)** stage and click the **Clone** icon when it appears. Select the new stage and rename it to **Ring 1 (Early Adopters)**.

Stages | + Add ▾



9. Select the **Ring 1 (Early Adopters)** stage from the Tasks drop-down list and select the **Azure CLI** task. Modify the script by changing the value from **10** to **30** to cause 30% of the traffic to go to the new feature site.

Inline Script \* ⓘ

```
az webapp traffic-routing set --resource-group $(ResourceGroupName) --name $(WebsiteName) --distribution staging=30
```

```
az webapp traffic-routing set --resource-group $(ResourceGroupName) --name $(WebsiteName) --distribution staging=30
```

It allows us to move the new feature into broader distribution if it works ok in smaller users.

10. From the menu above the tasks, click **Pipeline** to return to editing the release pipeline. Hover over the **Ring 1 (Early Adopters)** stage and when the **Clone** icon appears, click it. Click to select the new stage and rename it to **Public**. Click **Save** and **OK**.

Stages | + Add ▾



11. Click the **Pre-deployment conditions** icon for the **Ring 1 (Early Adopters)** stage and add yourself as a pre-deployment approver. Do the same for the **Public** stage—Click **Save** and **OK**.

Stages | + Add ▾



The first step in releasing the new code to the public is to swap the new feature site (that is, the staging site) with the production so that production is now running the new code.

12. From the **Tasks** drop-down list, select the **Public** stage. Select the **Azure CLI** task, change the **Display name** to **Swap sites** and change the **Inline Script** to the following command:

```
az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName)  
--slot staging --target-slot production
```

Inline Script \* ⓘ

```
az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName) --slot staging --target-slot production
```

```
az webapp deployment slot swap -g $(ResourceGroupName) -n  
$(WebsiteName) --slot staging --target-slot production
```

Next, we need to remove any traffic from the staging site.

13. Right-click the **Swap sites** task and click **Clone tasks(s)**. Select the **Swap sites copy** task, change its **Display name** to **Stop staging traffic**, and set the **Inline Script** to the following:

```
az webapp traffic-routing set --resource-group $(ResourceGroupName) --name  
$(WebsiteName) --distribution staging=0
```

1. Click **Save**, then **OK** to save your work.
2. Click **Create release** and **Create** to start a release. Click the release link to see the progress.

## All pipelines > Ring-based Deployment

Release [Release-1](#) has been created

Pipeline Tasks Variables Retention Options History

Wait until Ring 0 (Canary) has succeeded.

Stages

```
graph LR; R0[Ring 0 (Canary)] --- R1((Ring 1 (Early Adopter))); R1 --- P[Public]; R0 -- "Succeeded" --> R1; R1 -- "Pending approval" --> P;
```

Ring 0 (Canary)  
Succeeded  
on 04/09/2019, 17:00

Ring 1 (Early Adopter)  
Pending approval

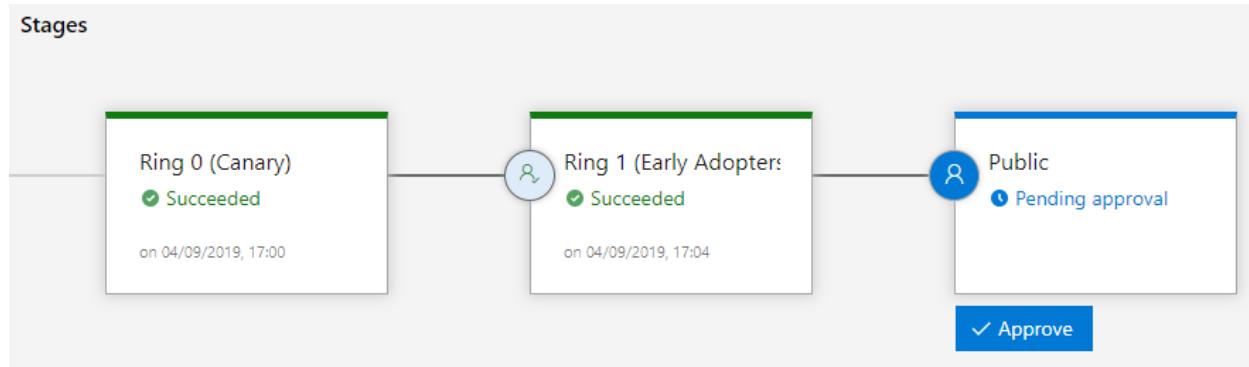
Public  
Not deployed

✓ Approve

Currently, 10% of the traffic will go to the new feature site.

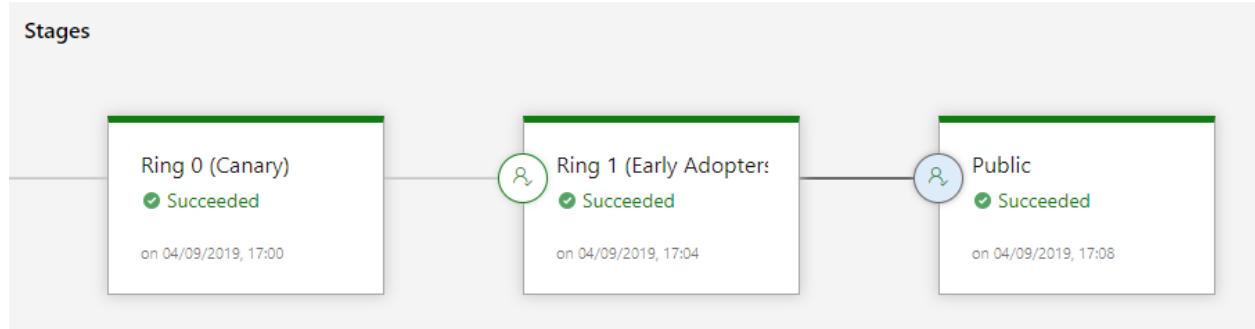
- Click **Approve** on the **Ring 1 (Early Adopters)** stage, and then **Approve**.

When this stage completes, 30% of the traffic will go to the early adopters in ring 1.



- Click **Approve** on the **Public** stage, and then **Approve**.

When this stage completes, all the traffic will go to the swapped production site, running the new code.



The new feature has been fully deployed.

# Implement canary releases and dark launching

## Explore canary releases

Completed 100 XP

- 1 minute

The term canary release comes from the days that miners took a canary with them into the coal mines.

The purpose of the canary was to identify the existence of toxic gasses.

The canary would die much sooner than the miner, giving them enough time to escape the potentially lethal environment.

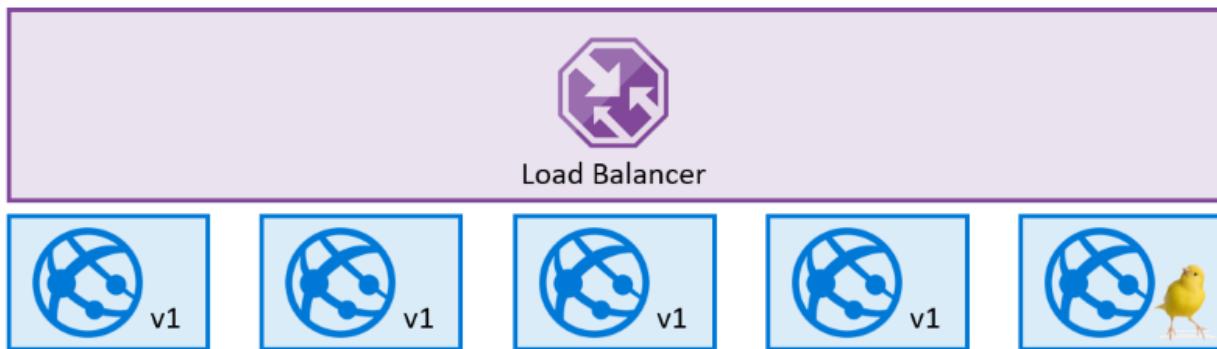
A canary release is a way to identify potential problems without exposing all your end users to the issue at once.

The idea is that you tell a new feature only to a minimal subset of users.

By closely monitoring what happens when you enable the feature, you can get relevant information from this set of users and either continue or rollback (disable the feature).

If the canary release shows potential performance or scalability problems, you can build a fix for that and apply that in the canary environment.

After the canary release has proven to be stable, you can move the canary release to the actual production environment.



Canary releases can be implemented using a combination of feature toggles, traffic routing, and deployment slots.

- You can route a percentage of traffic to a deployment slot with the new feature enabled.
- You can target a specific user segment by using feature toggles.

## Examine Traffic Manager

Completed 100 XP

- 3 minutes

In the previous module, we saw how Deployment slots in Azure Web Apps enable you to swap between two different versions of your application quickly.

Suppose you want more control over the traffic that flows to your other versions. Deployment slots alone aren't enough.

To control traffic in Azure, you can use a component called Azure Traffic Manager.

## Azure Traffic Manager

Azure Traffic Manager is a DNS-based traffic load balancer that enables you to distribute traffic optimally to services across global Azure regions while providing high availability and responsiveness.

Traffic Manager uses DNS to direct client requests to the most appropriate service endpoint based on a traffic-routing method and the health of the endpoints.

An endpoint is an Internet-facing service hosted inside or outside of Azure.

Traffic Manager provides a range of traffic-routing methods and endpoint monitoring options to suit different application needs and automatic failover models.

Traffic Manager is resilient to failure, including the breakdown of an entire Azure region.

While the available options can change over time, the Traffic Manager currently provides six options to distribute traffic:

- **Priority:** Select Priority when you want to use a primary service endpoint for all traffic and provide backups if the primary or the backup endpoints are unavailable.
- **Weighted:** Select Weighted when you want to distribute traffic across a set of endpoints, either evenly or according to weights, which you define.
- **Performance:** Select Performance when you have endpoints in different geographic locations, and you want end users to use the "closest" endpoint for the lowest network latency.
- **Geographic:** Select Geographic so that users are directed to specific endpoints (Azure, External, or Nested) based on which geographic location their DNS query originates from. It empowers Traffic Manager customers to enable scenarios where knowing a user's geographic region and routing them based on that is necessary. Examples include following data sovereignty mandates, localization of content & user experience, and measuring traffic from different regions.
- **Multivalue:** Select MultiValue for Traffic Manager profiles that can only have IPv4/IPv6 addresses as endpoints. When a query is received for this profile, all healthy endpoints are returned.
- **Subnet:** Select the Subnet traffic-routing method to map sets of end-user IP address ranges to a specific endpoint within a Traffic Manager profile. The endpoint returned will be mapped for that request's source IP address when a request is received.

When we look at the options the Traffic Manager offers, the most used option for Continuous Delivery is routing traffic based on weights.

### Note

Traffic is only routed to endpoints that are currently available.

For more information, see also:

- [What is Traffic Manager?](#)
- [How Traffic Manager works](#)
- [Traffic Manager Routing Methods](#)

## Controlling your canary release

Using a combination of feature toggles, deployment slots, and Traffic Manager, you can achieve complete control over the traffic flow and enable your canary release.

You deploy the new feature to the new deployment slot or a new instance of an application, and you enable the feature after verifying the deployment was successful.

Next, you set the traffic to be distributed to a small percentage of the users.

You carefully watch the application's behavior, for example, by using application insights to monitor the performance and stability of the application.

## Understand dark launching

Completed 100 XP

- 1 minute

Dark launching is in many ways like canary releases.

However, the difference here's that you're looking to assess users' responses 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're being used as test users for the new feature, and often you don't even highlight the new feature to them, as such the term "Dark" launching.

Another example of dark launching is launching a new feature and using it on the backend to get metrics.

Let me illustrate with a real-world "launch" example.

As Elon Musk describes in his biography, they apply all kinds of Agile development principles in SpaceX.

SpaceX builds and launches rockets to launch satellites. SpaceX also uses dark launching.

When they have a new version of a sensor, they install it alongside the old one.

All data is measured and gathered both by the old and the new sensor.

Afterward, they compare the outcomes of both sensors.

Only when the new one has the same or improved results the old sensor is replaced.

The same concept can be applied to software. You run all data and calculations through your new feature, but it isn't "exposed" yet.

## How to implement dark launching

In essence, dark launching doesn't differ from a canary release or the implementation and switching of a feature toggle.

The feature is released and only exposed at a particular time.

As such, the techniques, as described in the previous chapters, do also apply for dark launching.

# Implement blue-green deployment and feature toggles

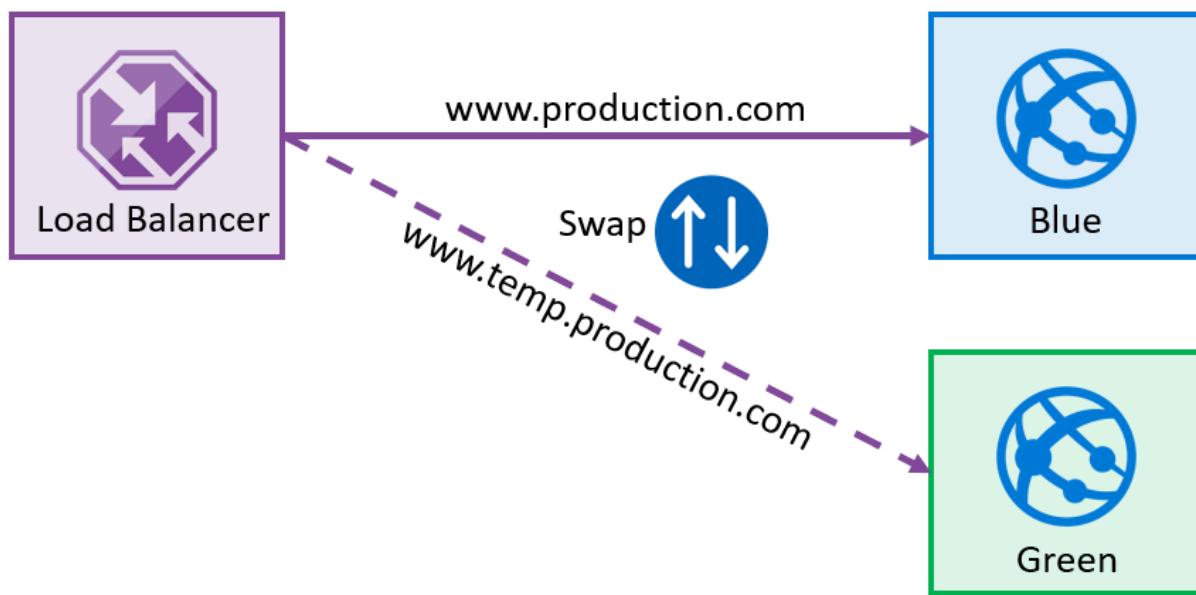
## What is blue-green deployment?

Completed 100 XP

- 3 minutes

Blue-green deployment is a technique that reduces risk and downtime by running two identical environments. These environments are called blue and green.

Only one of the environments is live, with the live environment serving all production traffic.



For this example, blue is currently live, and green is idle.

As you prepare a new version of your software, the deployment and final testing stage occur in an environment that isn't live: in this example, green. Once you've deployed and thoroughly tested the software in green, switch the router or load balancer so all incoming requests go to green instead of blue.

Green is now live, and blue is idle.

This technique can eliminate downtime because of app deployment. Besides, blue-green deployment reduces risk: if something unexpected happens with your new version on the green, you can immediately roll back to the last version by switching back to blue.

When it involves database schema changes, this process isn't straightforward. You probably can't swap your application. In that case, your application and architecture should be built to handle both the old and the new database schema.

## Explore deployment slots

Completed 100 XP

- 3 minutes

When using a cloud platform like Azure, doing blue-green deployments is relatively easy. You don't need to write your code or set up infrastructure. You can use an out-of-the-box feature called deployment slots when using web apps.

Deployment slots are a feature of Azure App Service. They're live apps with their hostnames. You can create different slots for your application (for example, Dev, Test, or Stage). The production slot is the slot where your live app stays. You can validate app changes in staging with deployment slots before swapping them with your production slot.

You can use a deployment slot to set up a new version of your application, and when ready, swap the production environment with the new staging environment. It's done by an internal swapping of the IP addresses of both slots.

## Swap

The swap eliminates downtime when you deploy your app with seamless traffic redirection, and no requests are dropped because of swap operations.

To learn more about Deployment slots and swap, see also:

- Set up Staging Environments in Azure App Service.
- Considerations on using Deployment Slots in your DevOps Pipeline.
- What happens during a swap.

# Exercise - set up a blue-green deployment

Completed 100 XP

- 60 minutes

In this demonstration, you'll investigate Blue-Green Deployment.

## Steps

Let's now look at how a release pipeline can be used to implement blue-green deployments.

We'll start by creating a new project with a release pipeline that can deploy the **Parts Unlimited** template again.

## An initial app deployment

1. Navigate to Azure DevOps Demo Generator in a browser: <https://azureddevopsdemogenerator.azurewebsites.net> and click **Sign in**.  
You'll be prompted to sign in if necessary.
2. In the **Create New Project** window, select your existing Organization, set the **Project Name** to **PU Hosted**, and click **Choose template**.

The screenshot shows the 'Choose a template' page of the Azure DevOps Demo Generator. At the top, there is a blue header bar with tabs for 'General', 'DevOps Labs', 'Microsoft Learn', and 'Microsoft Cloud Adoption Framework'. Below the header, there are six project templates displayed in a grid:

- Tailwind Traders**: An ASP.NET & React application using Azure App Service, AKS, Cosmos DB, Logic App, and Function App. Tags: Agile, Data and AI, React.
- SmartHotel360**: A reference sample app with several consumer and line-of-business apps and an Azure backend. Tags: scrum, aspnetcore, azureappservice.
- MyHealthClinic**: A scrum-based team project for an ASP.NET Core web application. Tags: scrum, aspnetcore, azureappservice.
- PartsUnlimited**: A scrum-based team project using Azure DevOps. Tags: scrum, aspnetcore, azureappservice.
- PartsUnlimited-YAML**: A scrum-based team project using YAML definitions. Tags: scrum, aspnetcore, azureappservice.
- MyShuttle**: A Java application using Azure DevOps. Tags: scrum, java, application, azure web app, mysql.

At the bottom right of the page is a blue button labeled 'Select Template'.

3. Click on the **PartsUnlimited** project (not the PartsUnlimited-YAML project), click **Select Template**, and click **Create Project**. When the deployment completes, click **Navigate to the project**.
4. In the main menu for **PU Hosted**, click **Pipelines**, then click **Builds**, then **Queue**, and finally **Run** to start a build.

The build should succeed.

### Note

*Warnings might appear but can be ignored for this walkthrough.*

 #20190904.1: Updated FullEnvironmentSetupMerged.param.json  
Manually run today at 12:26 by Greg Low ⚡ PartsUnlimited ➔ master ⚡ 58d9b87

[Logs](#) [Summary](#) [Tests](#) [WhiteSource Bolt Build Report](#)

---

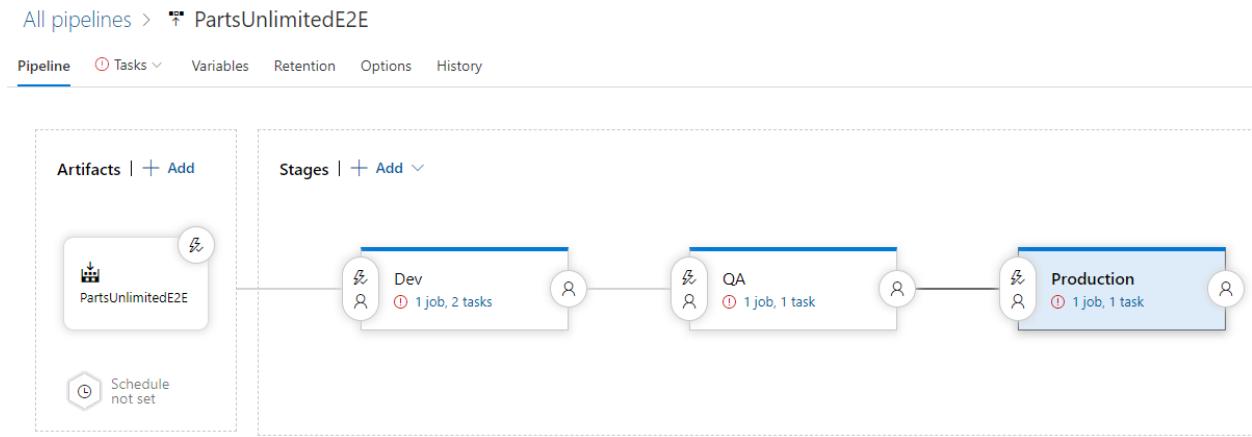
### Phase 1

Pool: [Azure Pipelines](#) · Agent: Hosted Agent

---

|   |                        |
|---|------------------------|
|  Prepare job         | · succeeded            |
|  Initialize job      | · succeeded            |
|  Checkout            | · succeeded            |
|  NuGet restore       | · succeeded 3 warnings |
|  Build solution      | · succeeded 1 warning  |
|  Test Assemblies     | · succeeded 1 warning  |
|  Copy Files          | · succeeded            |
|  Publish Artifact    | · succeeded            |
|  Post-job: Checkout  | · succeeded            |
|  Finalize Job        | · succeeded            |
|  Report build status | · succeeded            |

5. In the main menu, click **Releases**. Because a continuous integration trigger was in place, a release was attempted. However, we haven't yet configured the release so it will have failed. Click **Edit** to enter edit mode for the release.



6. Select the Dev stage from the drop-down list beside **Tasks**, then click to select the **Azure Deployment** task.
7. In the **Azure resource group deployment** pane, select your Azure subscription, then click **Authorize** when prompted. When authorization completes, choose a **Location** for the web app.

#### Note

*You might be prompted to sign in to Azure at this point.*

### Azure resource group deployment i

View YAML Remove

Task version 2.\* ▼

Display name \*

Azure Deployment

Azure Details ^

Azure subscription \* (i) | [Manage](#) ↳

Microsoft

(i) Scoped to subscription 'Microsoft Azure Sponsorship'

Action \* (i)

Create or update resource group

Resource group \* (i)

`$(ResourceGroupName)`

Location \* (i)

East US

Template ^

Template location \*

Linked artifact

Template \* (i)

`$(System.DefaultWorkingDirectory)/PartsUnlimitedE2E/drop/PartsUnlimited-aspnet45/env/PartsUnlimitedEnv/Templates/FullEnvironmentSetupMerged.json`

...

8. Click **Azure App Service Deploy** in the task list to open its settings. Again, select your Azure subscription. Set the **Deployment slot** to **Staging**.

## Azure App Service deploy ⓘ

Task version 3.\*

Display name \*

Azure App Service Deploy

Azure subscription \* ⓘ | Manage ↗

Microsoft

Scoped to subscription 'Microsoft Azure Sponsorship'

App type \* ⓘ

Web App

App Service name \* ⓘ

\$(WebsiteName)

Deploy to slot ⓘ

Resource group \* ⓘ

\$(ResourceGroupName)

Slot \* ⓘ

Staging

### Note

The template creates a production site and two deployment slots: Dev and Staging. We'll use Staging for our Green site.

9. In the task list, click **Dev**, and in the **Agent job** pane, select **Azure Pipelines** for the **Agent pool** and **windows-latest** for the **Agent Specification**.

## Agent job ⓘ

Display name \*

Dev

Agent selection ^

Agent pool ⓘ | Pool information | Manage ↗

Azure Pipelines

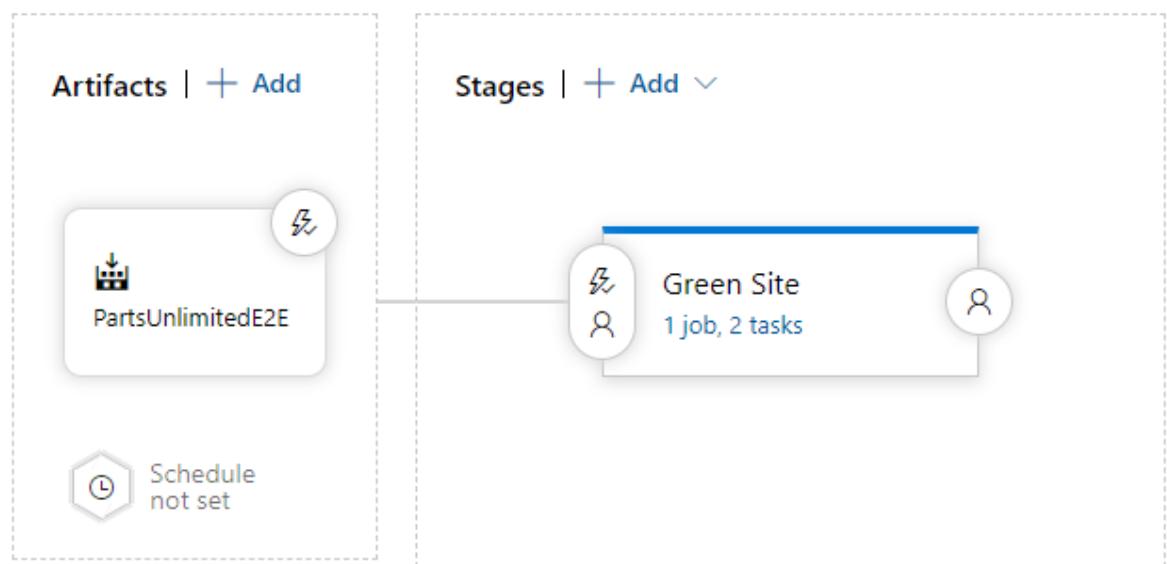
Agent Specification \*

vs2017-win2016

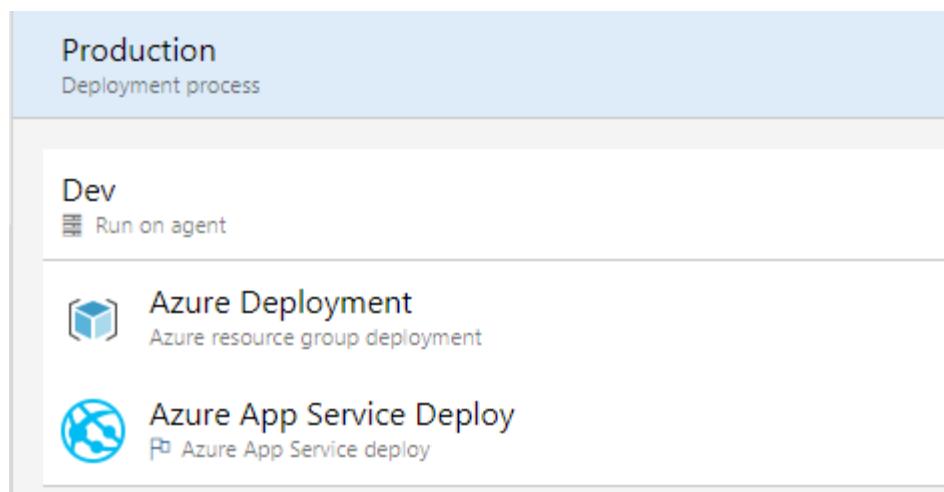
10. From the top menu, click **Pipelines**. Click the **Dev** stage, and in the properties window, rename it to **Green Site**. Click the **QA** stage and click **Delete** and **Confirm**. Click the **Production** stage and click **Delete** and **Confirm**. Click **Save**, then **OK**.

All pipelines > PartsUnlimitedE2E

Pipeline Tasks Variables Retention Options History



11. Hover over the **Green Site** stage and click the **Clone** icon when it appears. Change the **Stage name** to **Production**. From the **Tasks** drop-down list, select **Production**.



12. Click the **Azure App Service Deploy** task and uncheck the **Deploy to slot** option. Click **Save** and **OK**.

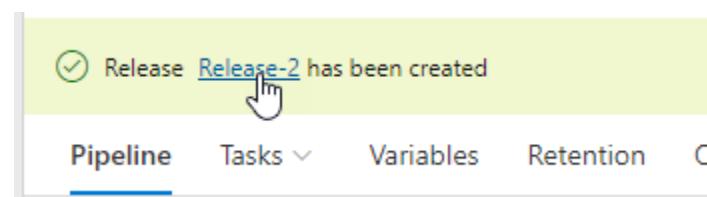
App Service name \* (i)

Deploy to slot (i)

Virtual application (i)

The production site isn't deployed to a deployment slot. It's deployed to the main site.

13. Click **Create release**, then **Create** to create the new release. When created, click the release link to view its status.



After a while, the deployment should succeed.

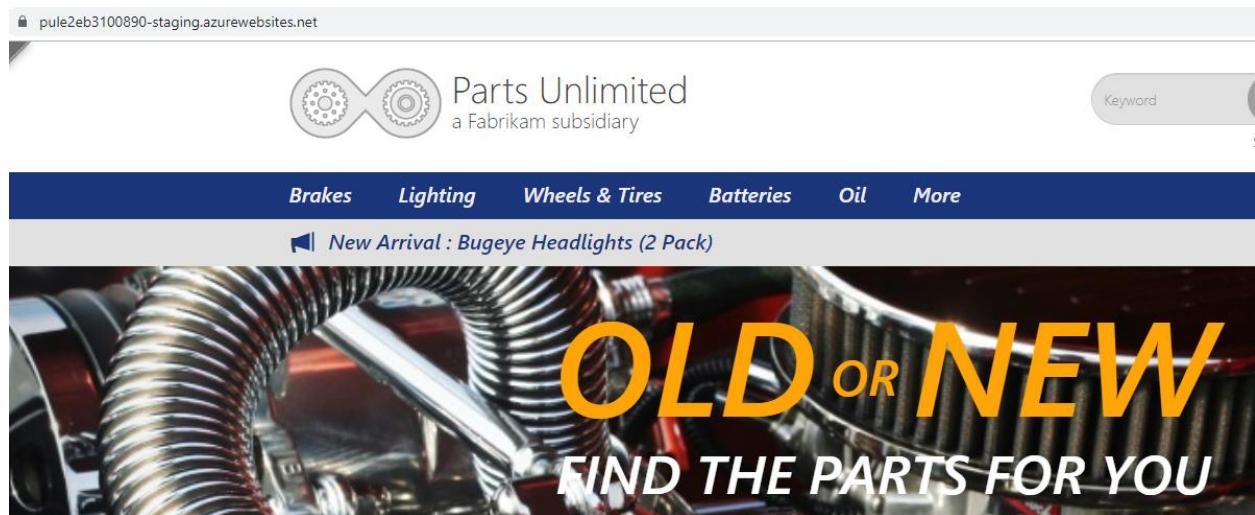
A screenshot of the Azure DevOps Release blade. On the left, under the "Release" section, it says "Manually triggered by Greg Low on 04/09/2019, 14:02". Below that, under "Artifacts", it shows "PartsUnlimitedE2E 20190904.1" with a "master" branch indicator. On the right, under the "Stages" section, there are two stages: "Green Site" and "Production". Both stages are shown as green boxes with a checkmark icon and the word "Succeeded". The "Green Site" stage was completed on 04/09/2019, 14:05, and the "Production" stage was completed on 04/09/2019, 14:09.

## Test the green site and the production site

14. Open the blade for the **ASPDOTNET** resource group created by the project deployment in the Azure portal. Notice the names of the web apps that have been deployed. Click to open the *Staging\** web app's blade. Copy the URL from the top left-hand side.

|  |   |
|--|---|
| Resource group ( <a href="#">change</a> ) : <b>ASPDOTNET</b> | URL : <a href="https://pule2eb3100890-staging.azurewebsites.net">https://pule2eb3100890-staging.azurewebsites.net</a> |
| Status : Running   | App Service Plan : <a href="#">pule2e (S1: 1)</a>   |
| Location : East US   | <a href="https://pule2eb3100890-staging.azurewebsites.net">https://pule2eb3100890-staging.azurewebsites.net</a>       |

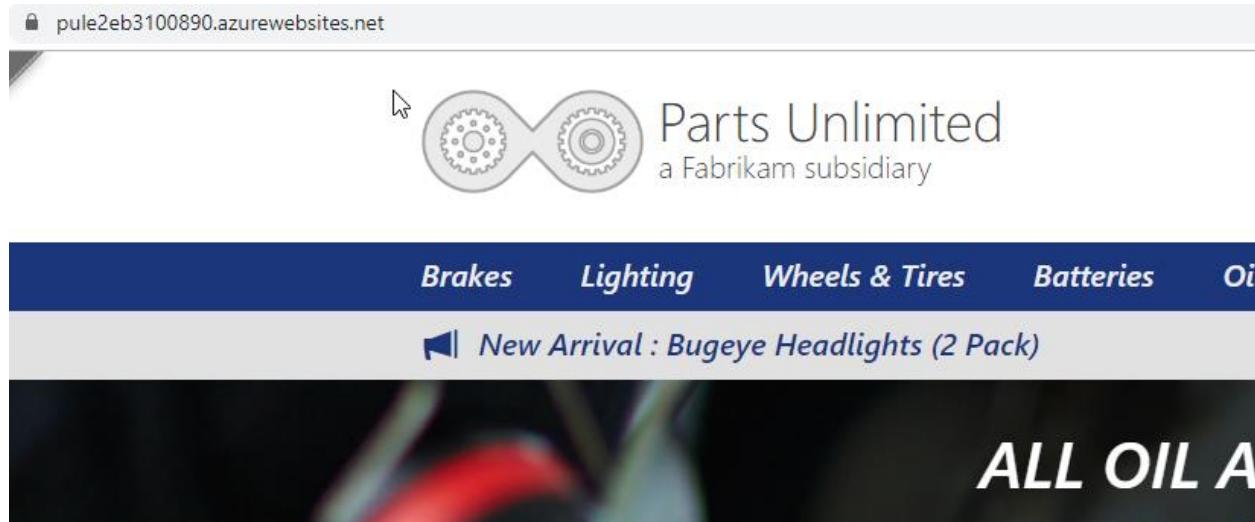
15. Open a new browser tab and navigate to the copied URL. It will take the application a short while to compile, but then the Green website (on the Staging slot) should appear.



### Note

You can tell that the staging slot is being used because of the **-staging** suffix in the website URL.

16. Open another new browser tab and navigate to the same URL but without the **-staging** slot. The production site should also be working.



### Note

Leave both browser windows open for later in the walkthrough.

## Configure blue-green swap and approval

Now that both sites are working, let's configure the release pipeline for blue-green deployment.

17. In **Azure DevOps**, in the main menu for the **PU Hosted** project, click **Pipelines**, then click **Releases**, then click **Edit** to return to edit mode.
18. Click the **Production** stage, click **Delete**, then **Confirm** to remove it. Click **+Add** to add an extra stage and click **Empty job** for the template. Set **Swap Blue-Green** for the **Stage name**.

Stages | + Add ▾



19. Click **Variables** and modify the **Scope** of **WebsiteName** to **Release**.

The screenshot shows the 'Variables' tab in the Azure DevOps interface. At the top, there are tabs for 'Variables', 'Retention', 'Options', and 'History'. Below the tabs is a search bar labeled 'Filter by keywords'. To the right of the search bar is a 'Scope' dropdown set to 'Release'. The main area is a table with columns 'Name', 'Value', and 'Scope'. There are four rows in the table:

| Name              | Value          | Scope   |
|-------------------|----------------|---------|
| HostingPlan       | pule2e         | Release |
| ResourceGroupName | ASPDOTNET      | Release |
| ServerName        | pule2eb3100890 | Release |
| WebsiteName       | pule2eb3100890 | Release |

20. From the **Tasks** drop-down list, click to select the **Swap Blue-Green** stage. Click the + to the right-hand side of **Agent Job** to add a new task. In the **Search** box, type **CLI**.

The screenshot shows the 'Tasks' search results. At the top left is a 'Add tasks' button and a 'Refresh' button. To the right is a search bar containing the text 'cli'. Below the search bar is a list of tasks:

- Docker CLI installer**: Install Docker CLI on agent machine.
- Azure CLI**: Run Azure CLI commands against an Azure subscription in a Shell script when running on Linux agent or Batch script when running on Windows agent.
- Python pip authenticate**: Authentication task for the pip client used for installing Python distributions

21. Hover over the **Azure CLI** template and when the **Add** button appears, click it, then click to select the **Azure CLI** task to open its settings pane.

Azure CLI ⓘ

Task version 1.\*

Display name \*

Azure CLI

Azure subscription \* ⓘ | Manage ↗

This setting is required.

Script Location \*

Script path

Script Path \* ⓘ

This setting is required.

Arguments ⓘ

...

...

22. Configure the pane as follows, with your subscription, a **Script Location** of **Inline script**, and the **Inline Script**:

```
Az webapp deployment slot swap -g $(ResourceGroupName) -n $(WebsiteName)  
--slot Staging --target-slot production
```

Azure CLI ⓘ

Task version 1.\*

Display name \*

Azure CLI

Azure subscription \* ⓘ | Manage ↗

Microsoft

Scoped to subscription 'Microsoft Azure Sponsorship'

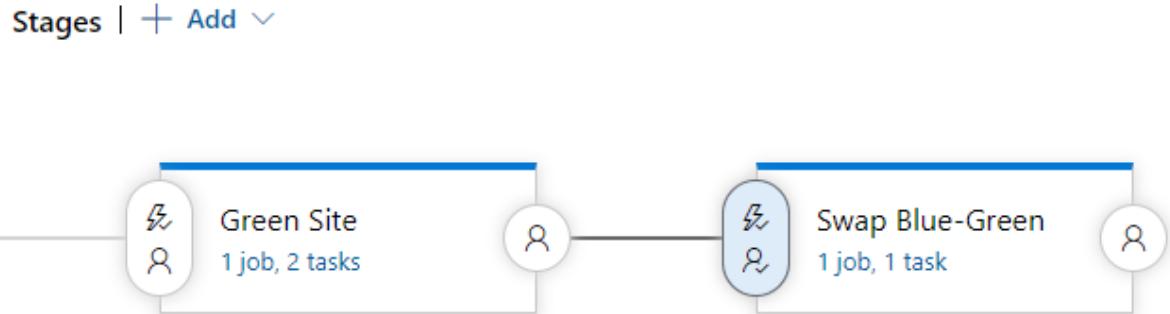
Script Location \*

Inline script

Inline Script \*

az webapp deployment slot swap -g \$(ResourceGroupName) -n \$(WebsiteName) --slot Staging --target-slot production

23. From the menu above the task list, click **Pipeline**. Click the **Pre-deployment conditions** icon for the **Swap Blue-Green** stage, then in the **Triggers** pane, enable **Pre-deployment approvals**.
24. Configure yourself as an approver, click **Save**, then **OK**.



## Test the blue-green swap

25. In the **PU Hosted** main menu, click **Repos**, then click **Files** to open the project files. Navigate to the following file.

```
master PartsUnlimited / PartsUnlimited-aspNet45 / src / PartsUnlimitedWebsite / Views / Home / Index.cshtml

Contents History Compare Blame Edit Rename Delete Download

1 @model PartsUnlimited.ViewModels.HomeViewModel
2
3 @{
4     ViewBag.Title = "Home Page";
5 }
6
7 @section scripts {
8 }
```

We'll make a cosmetic change to see that the website has been updated. We'll change the word **tires** in the main page rotation to **tyres** to target an international audience.

26. Click **Edit** to allow editing, then find the word **tires** and replace it with the word **tyres**. Click **Commit** and **Commit** to save the changes and trigger a build and release.

```

<div class="item" style="background-image: url('/Images/hero_imag
    <a href="@Url.Action("Browse", "Store", new { categoryId = 3
        <div class="container">
            <p>New tyres</p>
            <ul>
                <li>Improved fuel efficiency</li>
                <li>Superior wet weather breaking</li>
                <li>Added durability</li>
            </ul>
        </div>

```

27. From the main menu, click **Pipelines**, then **Builds**. Wait for the continuous integration build to complete successfully.

### PartsUnlimitedE2E

History Analytics

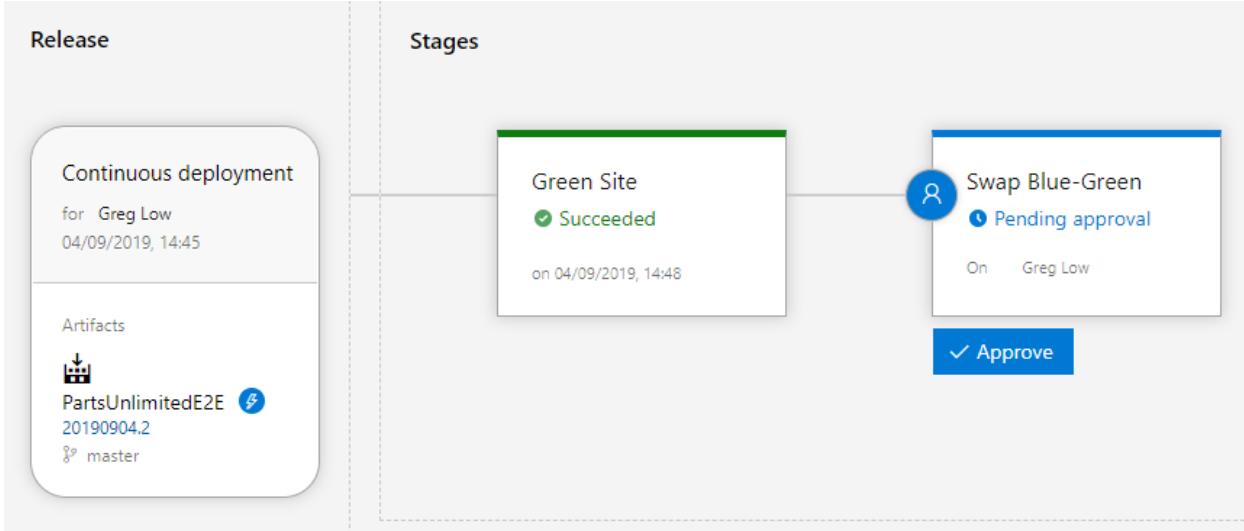
Commit

Build #

Branch

|  |  |  |
|--|--|--|
|  Updated Index.cshtml<br>CI build for Greg Low                              |  20190904.2   |  master |
|  Updated FullEnvironmentSetupMerged.param.json<br>Manual build for Greg Low |   20190904.1 |  master |

28. From the main menu, click **Releases**. Click to open the latest release (at the top of the list).



The screenshot shows the Azure DevOps Releases interface. On the left, under 'Release', there is a summary card for a 'Continuous deployment' for 'Greg Low' on '04/09/2019, 14:45'. Below this, under 'Artifacts', it lists 'PartsUnlimitedE2E' from '20190904.2' branch. To the right, under 'Stages', there are two cards: 'Green Site' which is marked as 'Succeeded' and completed on '04/09/2019, 14:48'; and 'Swap Blue-Green' which is in 'Pending approval' status. A blue button labeled 'Approve' is visible at the bottom of the swap stage card.

You're now being asked to approve the deployment swap across to Production. We'll check the green deployment first.

29. Refresh the Green site (that is, Staging slot) browser tab and see if your change has appeared. It now shows the altered word.

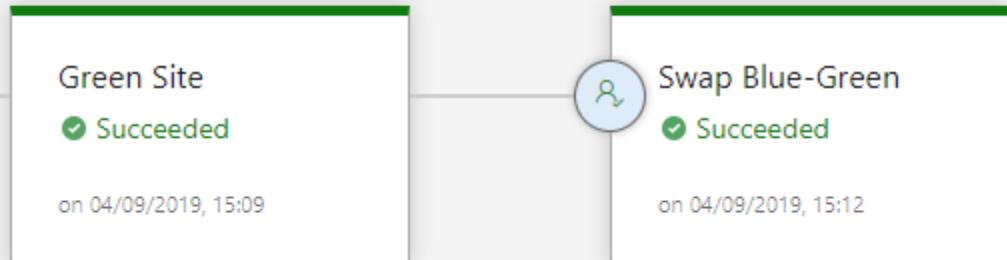


30. Refresh the Production site browser tab and notice that it still isn't updated.



31. As you're happy with the change, in release details, click **Approve**, then **Approve** and wait for the stage to complete.

## Stages



32. Refresh the Production site browser tab and check that it now has the updated code.



## Final notes

If you check the green site, you'll see it has the previous version of the code.

It's the critical difference with Swap, rather than just a typical deployment process from one staged site to another. You have a rapid fallback option by swapping the sites back if needed.

# Introduction to feature toggles

Completed 100 XP

- 3 minutes

Feature Flags allow you to change how our system works without making significant changes to the code. Only a small configuration change is required. In many cases, it will also only be for a few users.

Feature Flags offer a solution to the need to push new code into the trunk and deploy it, but it isn't functional yet.

They're commonly implemented as the value of variables used to control conditional logic.

Imagine that your team is all working in the main trunk branch of a banking application.

You've decided it's worth trying to have all the work done in the main branch to avoid messy operations of merge later.

Still, you need to ensure that significant changes to the interest calculations can happen, and people depend on that code every day.

Worse, the changes will take you weeks to complete. You can't leave the main code broken for that period.

A Feature Flag could help you get around it.

You can change the code so that other users who don't have the Feature Flag set will use the original interest calculation code.

The members of your team who are working on the new interest calculations and set to see the Feature Flag will have the new interest calculation code.

It's an example of a business feature flag used to determine business logic.

The other type of Feature Flag is a Release Flag. Now, imagine that after you complete the work on the interest calculation code, you're nervous about publishing a new code out to all users at once.

You have a group of users who are better at dealing with new code and issues if they arise, and these people are often called Canaries.

The name is based on the old use of the Canaries in coal mines.

You change the configuration so that the Canary users also have the Feature Flag set, and they'll start to test the new code as well. If problems occur, you can quickly disable the flag for them again.

Another release flag might be used for AB testing. Perhaps you want to find out if a new feature makes it faster for users to complete a task.

You could have half the users working with the original version of the code and the other half working with the new code version.

You can then directly compare the outcome and decide if the feature is worth keeping. Feature Flags are sometimes called Feature Toggles instead.

By exposing new features by just "flipping a switch" at runtime, we can deploy new software without exposing any new or changed functionality to the end-user.

The question is, what strategy do you want to use in releasing a feature to an end-user.

- Reveal the feature to a segment of users, so you can see how the new feature is received and used.
- Reveal the feature to a randomly selected percentage of users.
- Reveal the feature to all users at the same time.

The business owner plays a vital role in the process, and you need to work closely with them to choose the right strategy.

Just as in all the other deployment patterns mentioned in the introduction, the most crucial part is always looking at how the system behaves.

The idea of separating feature deployment from Feature exposure is compelling and something we want to incorporate in our Continuous Delivery practice.

It helps us with more stable releases and better ways to roll back when we run into issues when we have a new feature that produces problems.

We switch it off again and then create a hotfix. By separating deployments from revealing a feature, you make the opportunity to release a day anytime since the new software won't affect the system that already works.

## What are feature toggles?

Feature toggles are also known as feature flippers, feature flags, feature switches, conditional features, and so on.

Besides the power they give you on the business side, they also provide an advantage on the development side.

Feature toggles are a great alternative to branching as well. Branching is what we do in our version control system.

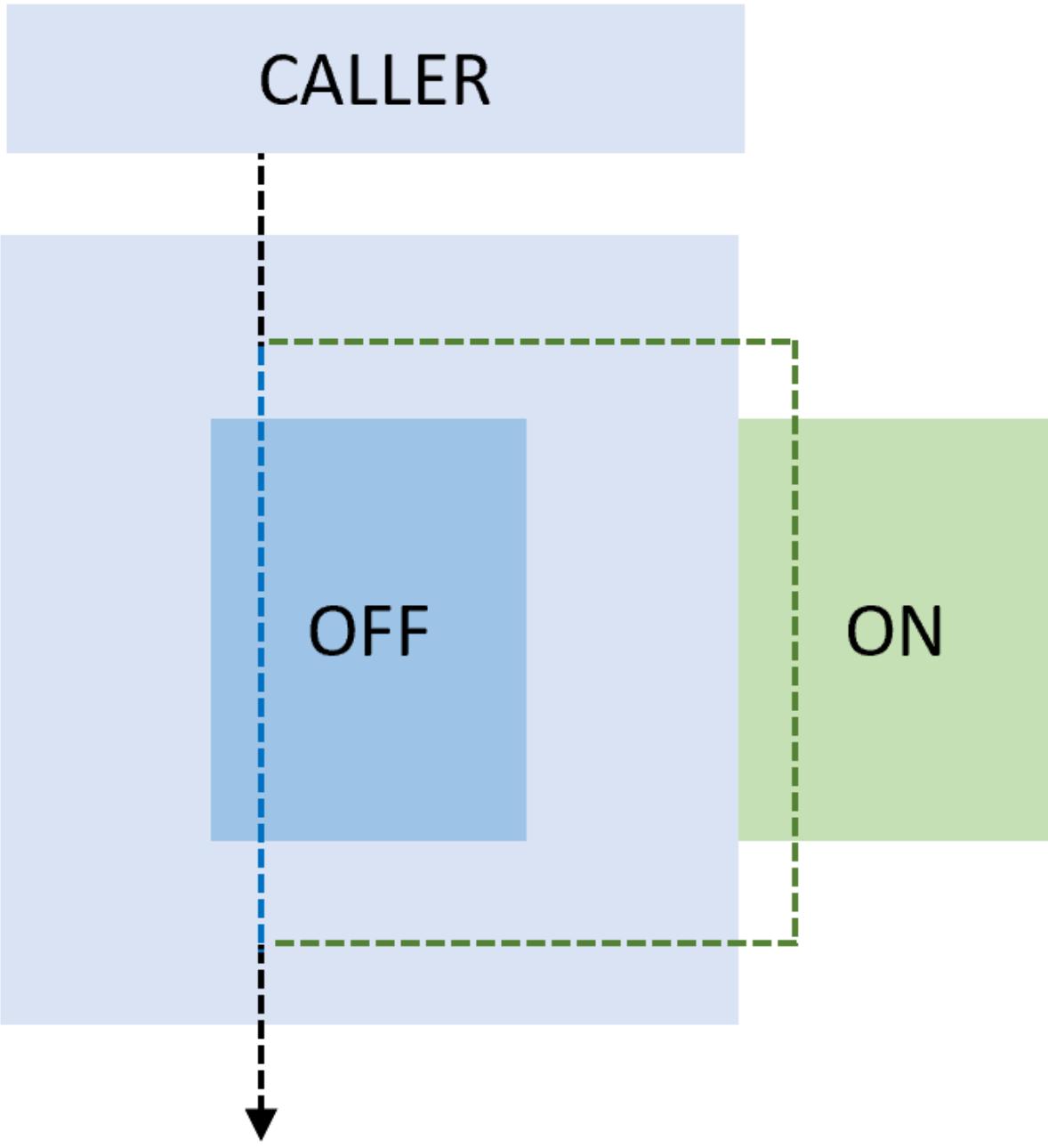
To keep features isolated, we maintain a separate branch.

When we want the software to be in production, we merge it with the release branch and deploy it.

With feature toggles, you build new features behind a toggle. Your feature is "off" when a release occurs and shouldn't be exposed to or impact the production software.

## How to implement a feature toggle

In the purest form, a feature toggle is an IF statement.



When the switch is off, it executes the code in the IF, otherwise the ELSE.

You can make it much more intelligent, controlling the feature toggles from a dashboard or building capabilities for roles, users, and so on.

If you want to implement feature toggles, many different frameworks are available commercially as Open Source.

For more information, see also [Explore how to progressively expose your features in production for some or all users.](#)

## Describe feature toggle maintenance

Completed 100 XP

- 2 minutes

A feature toggle is just code. And to be more specific, conditional code. It adds complexity to the code and increases the technical debt.

Be aware of that when you write them, and clean up when you don't need them anymore.

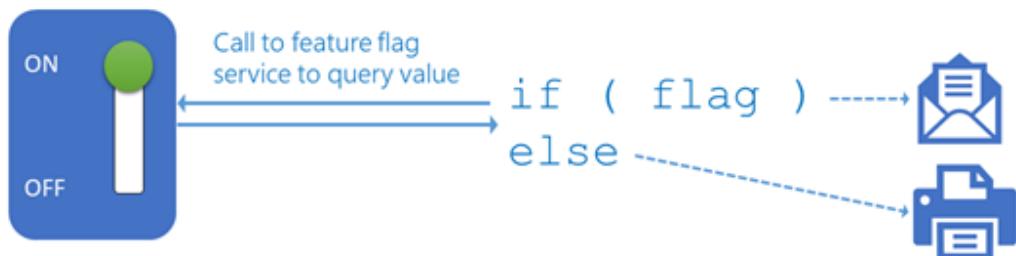
While feature flags can be helpful, they can also introduce many issues of their own.

The idea of a toggle is that it's short-lived and only stays in the software when it's necessary to release it to the customers.

You can classify the different types of toggles based on two dimensions as described by Martin Fowler.

He states that you can look at the dimension of how long a toggle should be in your codebase and, on the other side how dynamic the toggle needs to be.

## Planning feature flag lifecycles



The most important thing is to remember that you need to remove the toggles from the software.

If you don't do that, they'll become a form of technical debt if you keep them around for too long.

As soon as you introduce a feature flag, you've added to your overall technical debt.

Like other technical debt, they're easy to add, but the longer they're part of your code, the bigger the technical debt becomes because you've added scaffolding logic needed for the branching within the code.

The cyclomatic complexity of your code keeps increasing as you add more feature flags, as the number of possible paths through the code increases.

Using feature flags can make your code less solid and can also add these issues:

- The code is harder to test effectively as the number of logical combinations increases.
- The code is harder to maintain because it's more complex.
- The code might even be less secure.
- It can be harder to duplicate problems when they're found.

A plan for managing the lifecycle of feature flags is critical. As soon as you add a flag, you need to plan for when it will be removed.

Feature flags shouldn't be repurposed. There have been high-profile failures because teams decided to reuse an old flag that they thought was no longer part of the code for a new purpose.

## Tooling for release flag management

The amount of effort required to manage feature flags shouldn't be underestimated. It's essential to consider using tooling that tracks:

- Which flags exist.
- Which flags are enabled in which environments, situations, or target customer categories.
- The plan for when the flags will be used in production.
- The plan for when the flags will be removed.

Using a feature flag management system lets you get the benefits of feature flags while minimizing the risk of increasing your technical debt too high.

Azure App Configuration offers a Feature Manager. See [Azure App Configuration Feature Manager](#).

# Explore infrastructure as code and configuration management

## Explore environment deployment

Completed 100 XP

- 3 minutes

Suppose you've ever received a middle-of-the-night emergency support call because of a crashed server.

In that case, you know the pain of searching through multiple spreadsheets or even your memory—to access the manual steps of setting up a new machine.

Then there's also the difficulty of keeping the development and production environments consistent.

An easier way to remove the possibility of human error when initializing machines is to use Infrastructure as Code.

## Manual deployment versus infrastructure as code

A common analogy for understanding the differences between manual deployment and infrastructure as code is the distinction between owning pets and owning cattle.

When you have pets, you name each one and regard them as individuals; if something terrible happens to one of your pets, you're inclined to care a lot.

If you have a herd of cattle, you might still name them, but you consider their herd.

In infrastructure terms, there might be severe implications with a manual deployment approach if a single machine crashes and you need to replace it (pets).

If you adopt infrastructure as a code approach, you can more easily provision another machine without adversely impacting your entire infrastructure (cattle) if a single machine goes down.

## Implementing infrastructure as code

With infrastructure as code, you capture your environment (or environments) in a text file (script or definition).

Your file might include any networks, servers, and other computing resources.

You can check the script or definition file into version control and then use it as the source for updating existing environments or creating new ones.

For example, you can add a new server by editing the text file and running the release pipeline rather than remoting it into the environment and manually provisioning a new server.

The following table lists the significant differences between manual deployment and

| Manual deployment  | Infrastructure as code                           |
|--|--|
| Snowflake servers.   | A consistent server between environments.        |
| Deployment steps vary by environment.                        | Environments are created or scaled easily.       |
| More verification steps and more elaborate manual processes. | Fully automated creation of environment Updates. |
| Increased documentation to account for differences.          | Transition to immutable infrastructure.          |
| Deployment on weekends to allow time to recover from errors. | Use blue/green deployments.                      |
| Slower release cadence to minimize pain and long weekends.   | Treat servers as cattle, not pets.               |

## Benefits of infrastructure as code

The following list is benefits of infrastructure as code:

- Promotes auditing by making it easier to trace what was deployed, when, and how. (In other words, it improves traceability.)
- Provides consistent environments from release to release.
- Greater consistency across development, test, and production environments.
- Automates scale-up and scale-out processes.
- Allows configurations to be version-controlled.
- Provides code review and unit-testing capabilities to help manage infrastructure changes.

- Uses *immutable* service processes, meaning if a change is needed to an environment, a new service is deployed, and the old one was taken down, it isn't updated.
- Allows *blue/green* deployments. This release methodology minimizes downtime where two identical environments exist—one is live, and the other isn't. Updates are applied to the server that isn't live. When testing is verified and complete, it's swapped with the different live servers. It becomes the new live environment, and the previous live environment is no longer the live.
- Treats infrastructure as a flexible resource that can be provisioned, de-provisioned, and reprovisioned as and when needed.

## Examine environment configuration

Completed 100 XP

- 3 minutes

**Configuration management** refers to automated configuration management, typically in version-controlled scripts, for an application and all the environments needed to support it.

Configuration management means lighter-weight, executable configurations that allow us to have configuration and environments as code.

For example, adding a new port to a Firewall could be done by editing a text file and running the release pipeline, not by remoting into the environment and manually adding the port.

### Note

The term *configuration as code* can also be used to mean configuration management. However, it isn't used as widely, and in some cases, infrastructure as code is used to describe both provisioning and configuring machines. The term *infrastructure as code* is also sometimes used to include *configuration as code*, but not vice versa.

## Manual configuration versus configuration as code

Manually managing the configuration of a single application and environment can be challenging.

The challenges are even more significant for managing multiple applications and environments across multiple servers.

Automated configuration, or treating configuration as code, can help with some of the manual configuration difficulties.

The following table lists the significant differences between manual configuration and configuration as code.

| Manual configuration  | Configuration as code   |
|---|---|
| Configuration bugs are challenging to identify.                       | Bugs are easily reproducible.   |
| Error-prone.  | Consistent configuration.   |
| More verification steps and more elaborate manual processes.          | Increase deployment cadence to reduce the amount of incremental change. |
| Increased documentation.  | Treat environment and configuration as executable documentation.        |
| Deployment on weekends to allow time to recover from errors.          |   |
| Slower release cadence to minimize the requirement for long weekends. |   |

## Benefits of configuration management

The following list is benefits of configuration management:

- Bugs are more easily reproduced, audit help, and improve traceability.
- Provides consistency across environments such as dev, test, and release.
- It increased deployment cadence.
- Less documentation is needed and needs to be maintained as all configuration is available in scripts.
- Enables automated scale-up and scale out.
- Allows version-controlled configuration.
- Helps detect and correct configuration drift.

- Provides code-review and unit-testing capabilities to help manage infrastructure changes.
- Treats infrastructure as a flexible resource.
- Promotes automation.

# Understand imperative versus declarative configuration

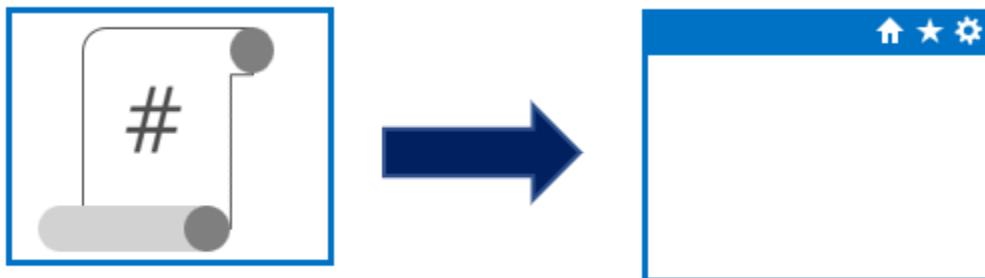
Completed 100 XP

- 3 minutes

There are a few different approaches that you can adopt to implement Infrastructure as Code and Configuration as Code.

Two of the main methods of approach are:

- **Declarative** (functional). The declarative approach states *what* the final state should be. When run, the script or definition will initialize or configure the machine to have the finished state declared without defining *how* that final state should be achieved.



- **Imperative** (procedural). In the imperative approach, the script states the *how* for the final state of the machine by executing the steps to get to the finished state. It defines what the final state needs to be but also includes how to achieve that final state. It also can consist of coding concepts such as *for*, *\*if-then*, *loops*, and *matrices*.



## Best practices

The **declarative** approach abstracts away the methodology of how a state is achieved. As such, it can be easier to read and understand what is being done.

It also makes it easier to write and define. Declarative approaches also separate the final desired state and the coding required to achieve that state.

So, it doesn't force you to use a particular approach, allowing for optimization.

A **declarative** approach would generally be the preferred option where ease of use is the primary goal. Azure Resource Manager template files are an example of a declarative automation approach.

An **imperative** approach may have some advantages in complex scenarios where changes in the environment occur relatively frequently, which need to be accounted for in your code.

There's no absolute on which is the best approach to take, and individual tools may be used in either *declarative* or *imperative* forms. The best approach for you to take will depend on your needs.

## Understand idempotent configuration

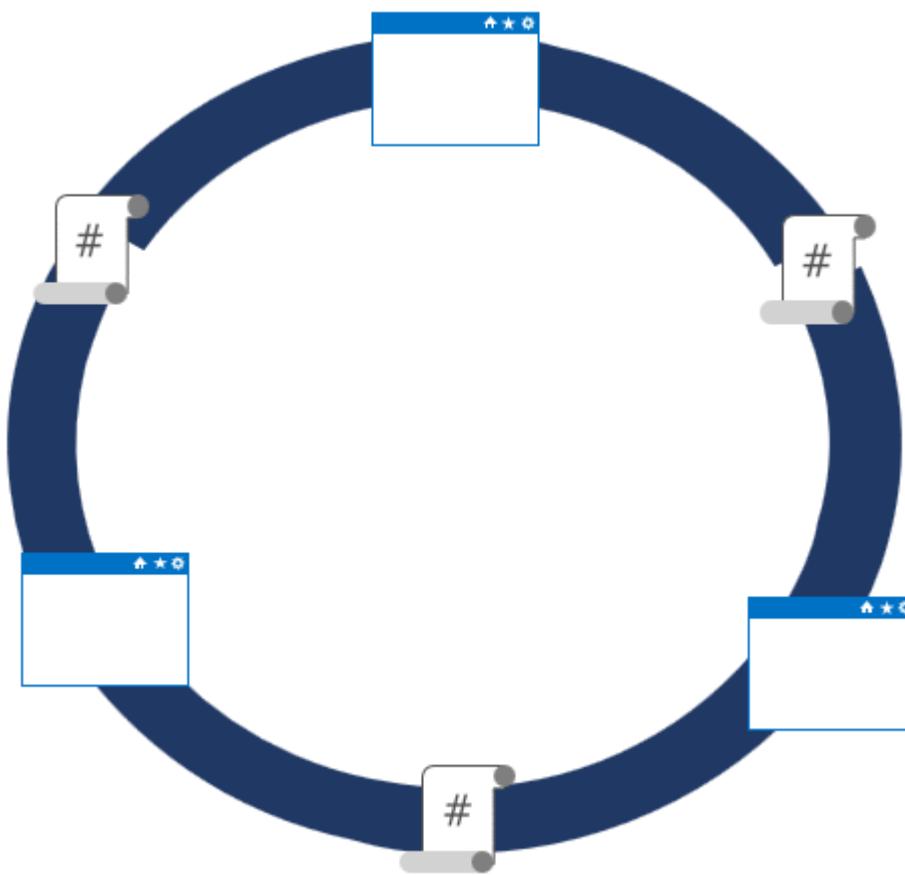
Completed 100 XP

- 3 minutes

**Idempotence** is a mathematical term that can be used in Infrastructure as Code and Configuration as Code. It can apply one or more operations against a resource, resulting in the same outcome.

For example, running a script on a system should have the same outcome despite the number of times you execute the script. It shouldn't error out or do the same actions irrespective of the environment's starting state.

In essence, if you apply a deployment to a set of resources 1,000 times, you should end up with the same result after each application of the script or template.



You can achieve idempotency by:

- Automatically configuring and reconfiguring an existing set of resources.
- Discarding the existing resources and recreating a new environment.

When defining Infrastructure as Code and Configuration as Code, as a best practice, build the scripts and templates in such a way as to embrace idempotency.

It's essential when working with cloud services because resources and applications can be scaled in and out regularly. New instances of services can be started up to provide service elasticity.

# Implement Bicep

## What is Bicep?

Completed 100 XP

- 2 minutes

**Azure Bicep** is the next revision of **ARM templates** designed to solve some of the issues developers were facing when deploying their resources to Azure. It's an Open Source tool and, in fact, a domain-specific language (DSL) that provides a means to declaratively codify infrastructure, which describes the topology of cloud resources such as VMs, Web Apps, and networking interfaces. It also encourages code reuse and modularity in designing the infrastructure as code files.

The new syntax allows you to write less code compared to ARM templates, which are more straightforward and concise and automatically manage the dependency between resources. Azure Bicep comes with its command line interface (CLI), which can be used independently or with Azure CLI. **Bicep CLI** allows you to *transpile* the Bicep files into ARM templates and deploy them and can be used to convert an existing ARM template to Bicep.

### Note

Beware that when converting ARM templates to Bicep, there might be issues since it's still a work in progress.

There's also an excellent integration with Visual Studio Code that creates a superb authoring experience. Azure Bicep supports types that are used to validate templates at development time rather than runtime. The extension also supports linting, which can be used to unify the development experience between team members or across different teams.

For more information about Azure Bicep, see [Bicep language for deploying Azure resources](#).

## Next steps

In the next unit, you'll find out various ways to install Bicep and set up your development environment.

# Install Bicep

Completed 100 XP

- 3 minutes

To start, install the Bicep CLI or the [Visual Studio Code Extension](#). Having both installed will give you a great authoring experience.

## Installing the VS Code extension

The extension provides language support, IntelliSense, and linting support.

The screenshot shows the Visual Studio Marketplace page for the Bicep extension. At the top, there's a navigation bar with the Microsoft logo and the text "Visual Studio | Marketplace". Below it, a breadcrumb navigation shows "Visual Studio Code > Programming Languages > Bicep". The main content area features a blue hexagonal icon with a white wrench and gear icon. To its right, the word "Bicep" is displayed in bold. Below the name, it says "Microsoft" with a blue checkmark icon, "379,797 installs", a 5-star rating with "(15)", and the word "Free". A brief description states "Bicep language support for Visual Studio Code". There are two buttons: a green "Install" button and a link "Trouble Installing?".

[Overview](#)

[Version History](#)

[Q & A](#)

[Rating & Review](#)

## Key features of the Bicep VS Code extension

The [Bicep VS Code extension](#) is capable of many of the features you would expect out of other language tooling. Here is a comprehensive list of the features that are currently implemented.

To verify you've installed it, create a file with the `.bicep` extension and watch the language mode change in the lower right corner of VS Code.

## Installing Bicep CLI

You need to have Azure CLI version **2.20.0** or later installed to be able to install the Bicep CLI. When you're ready, run the install command:

Azure CLICopy

```
az bicep install
```

You can upgrade the Bicep CLI by running the `az bicep upgrade`, and for validating the installation, use the `az bicep version`.

We deliberately avoided breaking down the installation for Windows, macOS, and Linux since Azure CLI is cross-platform, and the steps would be the same.

## Manual installation

You can manually install it if you don't have Azure CLI installed but still want to use Bicep CLI.

## Windows

You can use Chocolatey or Winget to install the Bicep CLI:

### Azure CLICopy

```
choco install bicep  
winget install -e --id Microsoft.Bicep  
bicep --help
```



```
bicep --version  
Bicep CLI version 0.4.626 (be3afce39f)
```

## Linux

To install the Bicep CLI on Linux manually, use the below script:

### BashCopy

```
curl -Lo bicep https://github.com/Azure/bicep/releases/latest/download/bicep-linux-x64  
chmod +x ./bicep  
sudo mv ./bicep /usr/local/bin/bicep  
bicep --help
```

## macOS

And for macOS, use homebrew or the previous script for Linux:

### BashCopy

```
brew tap azure/bicep  
brew install bicep  
bicep --version
```

## Troubleshooting the installation

If any errors or issues were faced during the installation, make sure you visit the [troubleshooting guide](#) on Azure Bicep documentation.

## Next steps

In the next unit, you'll create your first Bicep template and deploy it to Azure.

# Exercise - Create Bicep templates

Completed 100 XP

- 3 minutes

It's time to create your first Bicep template. After following this unit, you'll learn how the Bicep extension in VS Code simplifies development by providing type safety, syntax validation, and autocompletion.

## Prerequisites

To follow along, you'll need to have access to an Azure subscription. You also need to have:

- VS Code.
- Azure CLI.
- Bicep extension for VS Code.

## Create your first template

Open your VS Code and create a new file called `main.bicep`. When it's done, open the file and start typing `storage`. You should see a menu appears from which select the first option `res-storage` by pressing **Enter** or **Tab**. You should end up with a snippet that looks like:

BicepCopy

```
resource storageaccount 'Microsoft.Storage/storageAccounts@2021-02-01' = {
    name: 'name'
    location: location
    kind: 'StorageV2'
    sku: {
        name: 'Premium_LRS'
    }
}
```

This file will deploy an *Azure Storage Account*, however, we need to modify the file to make it ready for deployment. First let's add two parameters, one for the name since it should be unique, and one for the location.

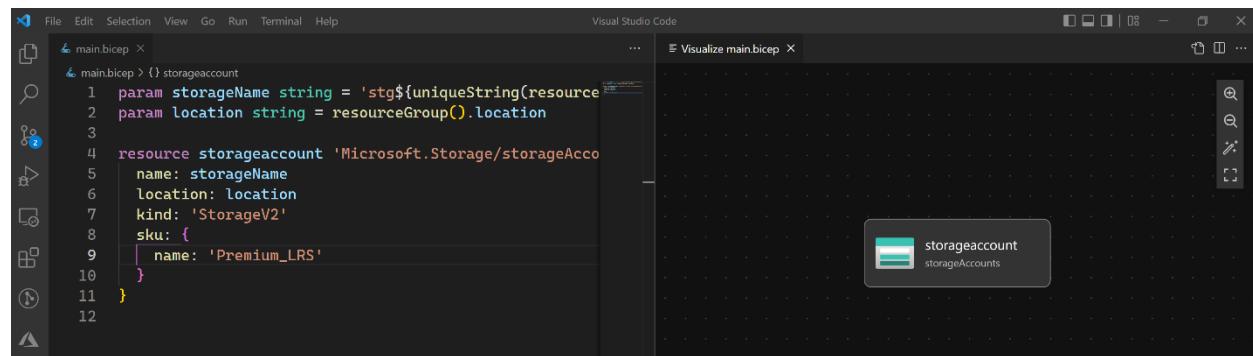
### BicepCopy

```
param storageName string = 'stg${uniqueString(resourceGroup().id)}'  
param location string = resourceGroup().location
```

The value you assign to the parameters is the default value that makes the parameters optional. Replace the name property with `storageName` and since the location is already used you're good to go ahead with the deployment.

## Visualize resources

You can use VS Code to visualize the resources defined in your Bicep file. Click on the visualizer button at the top right-hand corner:



## Compile the file

You don't need to compile the file to be able to deploy it, but it's good to know how you can do it.

Open the integrated terminal in VS Code by right-clicking on your Bicep file and selecting the **Open in Integrated Terminal** menu.

Use the build command as follows:

```
BashCopy  
az bicep build -f ./main.bicep
```

Feel free to take a look at the resulting ARM template and compare the two.

# Deploying the Bicep file

Now is the time to deploy the Bicep file you've created. In the same terminal run the following commands:

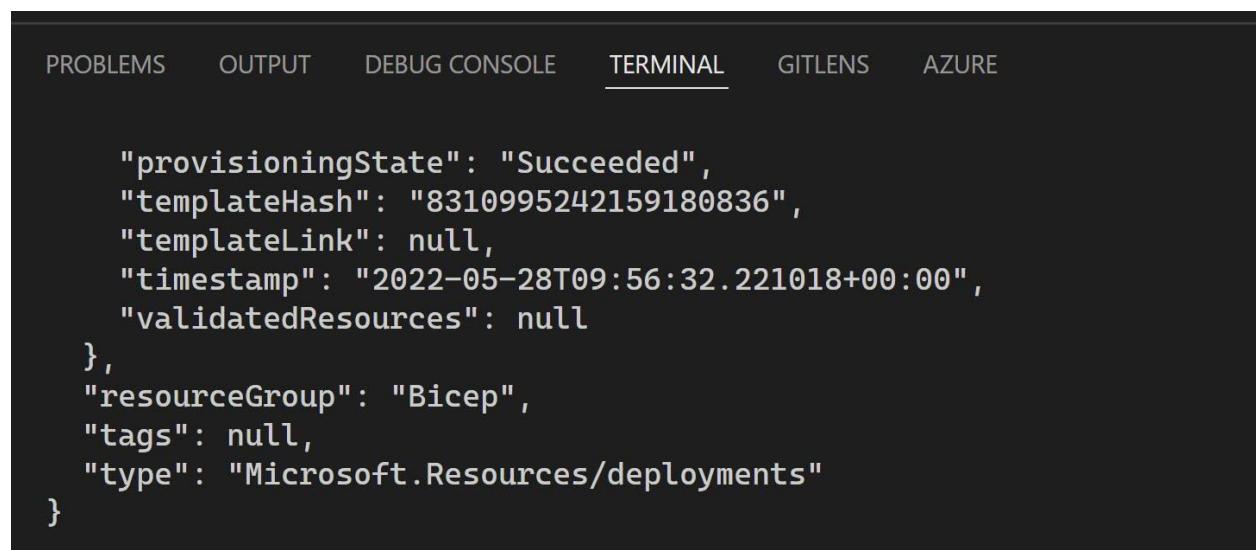
Azure CLICopy

```
az group create --name Bicep --location eastus
az deployment group create --resource-group Bicep --template-file main.bicep --
parameters storageName=uniqueName
```

## Note

Replace the `uniqueName` with a unique name, but you can also ignore providing the parameter since it has a default value.

When the deployment finishes, you'll be getting a message indicating the deployment succeeded.



```
"provisioningState": "Succeeded",
"templateHash": "8310995242159180836",
"templateLink": null,
"timestamp": "2022-05-28T09:56:32.221018+00:00",
"validatedResources": null
},
"resourceGroup": "Bicep",
"tags": null,
"type": "Microsoft.Resources/deployments"
}
```

## Next steps

Now that you've learned how to create a basic template and deploy it, head over to the next unit to learn more about the constructs in a Bicep file.

# Understand Bicep file structure and syntax

Completed 100 XP

- 4 minutes

Azure Bicep comes with its own syntax, however, it's easy to understand and follow. We won't go deep into the syntax and structure, but let's review the main concepts using an example.

## Sample Bicep .bicep file

BicepCopy

```
@minLength(3)
@maxLength(11)
param storagePrefix string

param storageSKU string = 'Standard_LRS'
param location string = resourceGroup().location

var uniqueStorageName = '${storagePrefix}${uniqueString(resourceGroup().id)}'

resource stg 'Microsoft.Storage/storageAccounts@2019-04-01' = {
    name: uniqueStorageName
    location: location
    sku: {
        name: storageSKU
    }
    kind: 'StorageV2'
    properties: {
        supportsHttpsTrafficOnly: true
    }
}

resource service 'fileServices' = {
    name: 'default'

        resource share 'shares' = {
            name: 'exampleshare'
        }
    }
}

module webModule './webApp.bicep' = {
    name: 'webDeploy'
    params: {
        skuName: 'S1'
        location: location
    }
}
```

```
        }  
    }  
  
output storageEndpoint object = stg.properties.primaryEndpoints
```

## Scope

By default the target scope of all templates is set for `resourceGroup`, however, you can customize it by setting it explicitly. As other allowed values, `subscription`, `managementGroup`, and `tenant`.

## Parameters

You've already used the parameters in the previous unit. They allow you to customize your template deployment at run time by providing potential values for names, location, prefixes, etc.

Parameters also have types that editors can validate and also can have default values to make them optional at deployment time. Additionally, you can see they can have validation rules to make the deployment more reliable by preventing any invalid value right from the authoring. For more information, see [Parameters in Bicep](#).

## Variables

Similar to parameters, variables play a role in making a more robust and readable template. Any complex expression can be stored in a variable and used throughout the template. When you define a variable, the type is inferred from the value.

In the above example, the `uniqueStorageName` is used to simplify the resource definition. For more information, see [Variables in Bicep](#).

## Resources

The `resource` keyword is used when you need to declare a resource in your templates. The resource declaration has a symbolic name for the resource that can be used to reference that resource later either for defining a subresource or for using its properties for an implicit dependency like a parent-child relationship.

There are certain properties that are common for all resources such as `location`, `name`, and `properties`. There are resource-specific properties that can be used to customize the resource pricing tier, `sku`, and so on.

You can define subresources within a resource or outside by referencing the parent. In the above example, a file share is defined within the storage account resource. If the intention was to define the resource outside of it, you would need to change your template:

### BicepCopy

```
resource storage 'Microsoft.Storage/storageAccounts@2021-02-01' = {
    name: 'examplestorage'
    location: resourceGroup().location
    kind: 'StorageV2'
    sku: {
        name: 'Standard_LRS'
    }
}

resource service 'Microsoft.Storage/storageAccounts/fileServices@2021-02-01' = {
    name: 'default'
    parent: storage
}

resource share 'Microsoft.Storage/storageAccounts/fileServices/shares@2021-02-01' = {
    name: 'exampleshare'
    parent: service
}
```

For more information, see [Resource declaration in Bicep](#).

## Modules

If you want truly reusable templates, you can't avoid using a module. Modules enable you to reuse a Bicep file in other Bicep files. In a module, you define what you need to deploy, and any parameters needed and when you reuse it in another file, all you need to do is reference the file and provide the parameters. The rest is taken care of by Azure Bicep.

In the above example, you're using a module that presumably is deploying an App Service. For more information, see [Using Modules in Bicep](#).

## Outputs

You can use outputs to pass values from your deployment to the outside world, whether it is within a CI/CD pipeline or in a local terminal or *Cloud Shell*. That would enable you to access a value such as storage endpoint or application URL after the deployment is finished.

All you need is the `output` keyword and the property you would like to access:

CloudCopy

```
output storageEndpoint endpoints = stg.properties.primaryEndpoints
```

To find more information, see [Outputs in Bicep](#).

## Other features

There are many other features available within a Bicep file such as loops, conditional deployment, multiline strings, referencing an existing cloud resource, and many more. In fact, any valid function within an ARM template is also valid within a Bicep file.

## Next steps

In the next unit, you'll learn how to use Bicep in an Azure Pipeline.

# Exercise - Deploy a Bicep file from Azure Pipelines

Completed 100 XP

- 3 minutes

Now that you know how to validate, compile and deploy your resources from your local environment, it's time to extend that and see how to bring that into an Azure Pipeline to streamline your deployment process even further.

## Prerequisites

You'll need an Azure Subscription, if you don't have one, [create a free account](#) before you begin.

You also need an Azure DevOps organization, similarly, if you don't have one, [create one for free](#).

You'll need to have a configured [service connection](#) in your project that is linked to your Azure subscription. Don't worry if you haven't done this before, we'll show you an easy way to do it when you're creating your pipeline.

You also need to have that Bicep file you created earlier pushed into the Azure Repository of your project.

## Creating the pipeline

1. From within your Azure DevOps project, select **Pipelines**, and **New pipeline**.
2. Select Azure Repos Git (YAML) and specify your Azure Repo as a source.

New pipeline

# Where is your code?



Azure Repos Git YAML

Free private Git repositories, pull requests, and code search

---



Bitbucket Cloud YAML

Hosted by Atlassian

---



GitHub YAML

Home to the world's largest community of developers

---



GitHub Enterprise Server YAML

The self-hosted version of GitHub Enterprise

---



Other Git

Any generic Git repository

---



Subversion

Centralized version control by Apache

3. Select the starter pipeline from the list of templates.

New pipeline

# Configure your pipeline



## Starter pipeline

Start with a minimal pipeline that you can customize to build and deploy your code.



## Existing Azure Pipelines YAML file

Select an Azure Pipelines YAML file in any branch of the repository.

Show more

4. Replace everything in the starter pipeline file with the following snippet.

```
BicepCopy
trigger:
  - main

name: Deploy Bicep files

variables:
  vmImageName: 'ubuntu-latest'

  azureServiceConnection: 'myServiceConnection'
  resourceName: 'Bicep'
  location: 'eastus'
  templateFile: 'main.bicep'

pool:
  vmImage: $(vmImageName)

steps:
  - task: AzureCLI@2
    inputs:
      azureSubscription: $(azureServiceConnection)
      scriptType: bash
      scriptLocation: inlineScript
      inlineScript: |
        az --version
        az group create --name $(resourceName) --location
$(location)
        az deployment group create --resource-group
$(resourceName) --template-file $(templateFile)
```

## Note

Don't forget to replace the service connection name with yours.

5. Select **Save and run** to create a new commit in your repository containing the pipeline YAML file and then run the pipeline. Wait for the pipeline to finish running and check the status.

The screenshot shows two side-by-side views. On the left is the 'Jobs in run #Deploy Bicep files' page from GitHub Actions, showing a single job with several steps: Initialize job, Checkout Bicep@main to s, AzureCLI, Post-job: Checkout Bicep@main..., Finalize Job, and Report build status. All steps are marked as completed with green checkmarks. The total duration was 55 seconds. On the right is the 'Job' log from Azure DevOps, which details the execution of the pipeline steps. The log includes information such as the pool (local), agent (me), start time, duration, and specific log entries for each step.

6. Once the pipeline runs successfully, you should be able to see the resource group and the storage account.

The screenshot shows the Azure portal interface for a resource group named 'Bicep'. The left sidebar lists various management options like Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, and Deployments. The main pane displays the 'Essentials' section under 'Resources'. It shows one record: a Storage account named 'stgk4xpvol5eofc' located in the 'Australia East' region. The table includes columns for Name, Type, Location, and other filtering options.

## Exercise - Deploy a Bicep file from GitHub workflows

Completed 100 XP

- 3 minutes

[GitHub Actions](#) are similar to Azure Pipelines in nature. They provide a way to automate software development and deployments. In this exercise, you'll learn how to deploy a Bicep file using a *GitHub Action*.

## Prerequisites

- You'll need a GitHub account that you can [create for free here](#).
- A GitHub repository is also required to store your Bicep file and workflows created earlier in the [Exercise - Create Bicep templates](#). Once you've created your GitHub repository, push the Bicep file into it.
- For deployment to Azure, access to an Azure subscription is needed, which can be [created for free here](#).

## Creating a service principal in Azure

To deploy your resources to Azure, you'll need to create a service principal which GitHub can use. So open a terminal or use Cloud Shell in the Azure portal and type the following commands:

CloudCopy

```
az login  
az ad sp create-for-rbac --name myApp --role contributor --scopes  
/subscriptions/{subscription-id}/resourceGroups/Bicep --sdk-auth
```

### Note

Don't forget to replace the Subscription ID with yours.

When the operation succeeds, it should output a JSON object that contains your `tenantId`, `subscriptionId`, `clientId`, `clientSecret`, and a few more properties, such as the following.

JSONCopy

```
{  
  "clientId": "<GUID>",  
  "clientSecret": "<GUID>",  
  "subscriptionId": "<GUID>",  
  "tenantId": "<GUID>",  
  (...)}
```

```
}
```

Note this object since you'll need to add it to your GitHub secrets.

## Creating a GitHub secret

In your GitHub repository, navigate to **Settings > Secrets > Actions**. Create a new secret called AZURE\_CREDENTIALS and paste the entire JSON object you got from the previous step.

Create another secret for the name of the resource group with a name such as AZURE\_RG and one for the subscription.

### Actions secrets

[New repository secret](#)

Secrets are environment variables that are **encrypted**. Anyone with **collaborator** access to this repository can use these secrets for Actions.

Secrets are not passed to workflows that are triggered by a pull request from a fork. [Learn more](#).

|  |                        |                        |                        |
|--|------------------------|------------------------|------------------------|
|  AZURE_CREDENTIALS  | Updated 16 minutes ago | <a href="#">Update</a> | <a href="#">Remove</a> |
|  AZURE_RG           | Updated 16 minutes ago | <a href="#">Update</a> | <a href="#">Remove</a> |
|  AZURE_SUBSCRIPTION | Updated now            | <a href="#">Update</a> | <a href="#">Remove</a> |

## Creating a GitHub action

1. First, navigate to your repository and select the Actions menu. Then, set up a workflow to create an empty workflow in your repository. You can rename the file to a different name if you prefer.



2. Replace the content of the file with the following snippet:

#### YAMLCopy

```
on: [push]
name: Azure ARM
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      # Checkout code
      - uses: actions/checkout@main

      # Log into Azure
      - uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      # Deploy Bicep file
      - name: deploy
        uses: azure/arm-deploy@v1
        with:
          subscriptionId: ${{ secrets.AZURE_SUBSCRIPTION }}
          resourceGroupName: ${{ secrets.AZURE_RG }}
          template: ./main.bicep
          parameters: storagePrefix=stg
          failOnStdErr: false
```

Feel free to replace the storage account prefix with your own.

#### Note

The first part of the workflow defines the trigger and its name. The rest defines a job and uses a few tasks to check out the code, sign in to Azure, and deploy the Bicep file.

3. Select **Start commit**, and enter a title and a description in the pop-up dialog. Then select **Commit directly to the main branch**, followed by **Commit a new file**.

Cancel changes

Start commit ▾

## Commit new file

Added a workflow

This is the initial workflow which deploys our Bicep file

- ⚡ Commit directly to the `main` branch.
- ↗ Create a **new branch** for this commit and start a pull request. [Learn more about pull requests.](#)

Commit new file

778

4. Navigate to the Actions tab and select the newly created action that should be running.

# main.yml

on: push

## ● build-and-deploy

5. Monitor the status and when the job is finished, check the Azure portal to see if the storage account is being created.

✔ Update main.yml Azure ARM #2

Re-run all jobs

...

The screenshot shows the Azure DevOps pipeline summary for the 'Update main.yml Azure ARM #2' run. The 'build-and-deploy' job is listed under 'Jobs' and is highlighted with a green checkmark icon. The job status is 'succeeded 30 minutes ago in 1m 29s'. The pipeline details show the following steps:

- > Set up job (4s)
- > Run actions/checkout@main (1s)
- > Run azure/login@v1 (6s)
- > deploy (1m 16s)
- > Post Run actions/checkout@main (0s)
- > Complete job (0s)

On the right side of the interface, there are buttons for 'Search logs', a refresh icon, and a gear icon for settings.

# Implement Desired State Configuration (DSC)

## Understand configuration drift

Completed 100 XP

- 2 minutes

**Configuration drift** is the process of a set of resources changing over time from their original deployment state.

It can be because of changes made manually by people or automatically by processes or programs.

Eventually, an environment can become a snowflake. A *snowflake* is a unique configuration that cannot be reproduced automatically and is typically a result of configuration drift.

With snowflakes, infrastructure administration and maintenance invariably involve manual processes, which can be hard to track and prone to human error.

The more an environment drifts from its original state, the more likely it is for an application to find issues.

The greater the degree of configuration drift, the longer it takes to troubleshoot and rectify issues.



## Security considerations

Configuration drift can also introduce security vulnerabilities into your environment. For example:

- Ports might be opened that should be kept closed.
- Updates and security patches might not be applied across environments consistently.
- The software might be installed that doesn't meet compliance requirements.

## Solutions for managing configuration drift

While eliminating configuration drift can be difficult, there are many ways you can manage it in your environments using configuration management tools and products such as:

- Windows PowerShell Desired State Configuration. It's a management platform in PowerShell that enables you to manage and enforce resource configurations. For more information about Windows PowerShell Desired State Configuration, go to [Windows PowerShell Desired State Configuration Overview](#).
- Azure Policy. Use Azure Policy to enforce policies and compliance standards for Azure resources. For more information about Azure Policy, go to [Azure Policy](#).

There are also other third-party solutions that you can integrate with Azure.

## Explore Desired State Configuration (DSC)

Completed 100 XP

- 3 minutes

**Desired State Configuration** (DSC) is a configuration management approach that you can use for configuration, deployment, and management of systems to ensure that an environment is maintained in a state that you specify (*defined state*) and doesn't deviate from that state.

DSC helps eliminate configuration drift and ensures the state is maintained for compliance, security, and performance.

Windows PowerShell DSC is a management platform in PowerShell that provides desired State.

PowerShell DSC lets you manage, deploy, and enforce configurations for physical or virtual machines, including Windows and Linux.

For more information, visit [Windows PowerShell Desired State Configuration Overview](#).

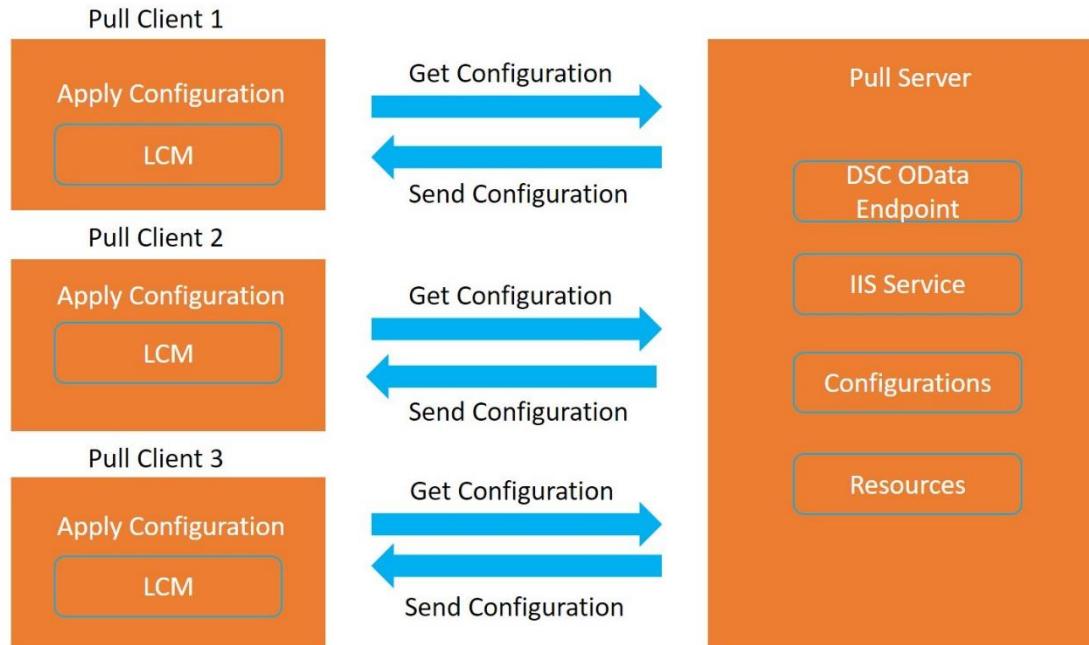
## DSC components

DSC consists of three primary parts:

- Configurations. These are declarative PowerShell scripts that define and configure instances of resources. Upon running the configuration, DSC (and the resources being called by the configuration) will apply the configuration, ensuring that the system exists in the state laid out by the configuration. DSC configurations are also *idempotent*: The Local Configuration Manager (LCM) will ensure that machines are configured in whatever state the configuration declares.
- Resources. They contain the code that puts and keeps the target of a configuration in the specified state. Resources are in PowerShell modules and can be written to a model as generic as a file or a Windows process or as specific as a Microsoft Internet Information Services (IIS) server or a VM running in Azure.
- Local Configuration Manager (LCM). The LCM runs on the nodes or machines you wish to configure. It's the engine by which DSC facilitates the interaction between resources and configurations. The LCM regularly polls the system using the control flow implemented by resources to maintain the state defined by a configuration. If the system is out of state, the LCM calls the code in resources to apply the configuration according to specified.

There are two methods of implementing DSC:

- Push mode - A user actively applies a configuration to a target node and pushes out the configuration.
- Pull mode is where pull clients are automatically configured to get their desired state configurations from a remote pull service. This remote pull service is provided by a *pull server* that acts as central control and manager for the configurations, ensures that nodes conform to the desired state, and reports on their compliance status. The pull server can be set up as an SMB-based pull server or an HTTPS-based server. HTTPS-based pull-server uses the Open Data Protocol (OData) with the OData Web service to communicate using REST APIs. It's the model we're most interested in, as it can be centrally managed and controlled. The following diagram provides an outline of the workflow of DSC pull mode.



# Explore Azure Automation State configuration (DSC)

Completed 100 XP

- 2 minutes

**Azure Automation State configuration DSC** is an Azure cloud-based implementation of PowerShell DSC, available as part of Azure Automation.

Azure Automation State configuration allows you to write, manage, and compile PowerShell DSC configurations, import DSC Resources, and assign configurations to target nodes, all in the cloud.

## Why use Azure Automation DSC?

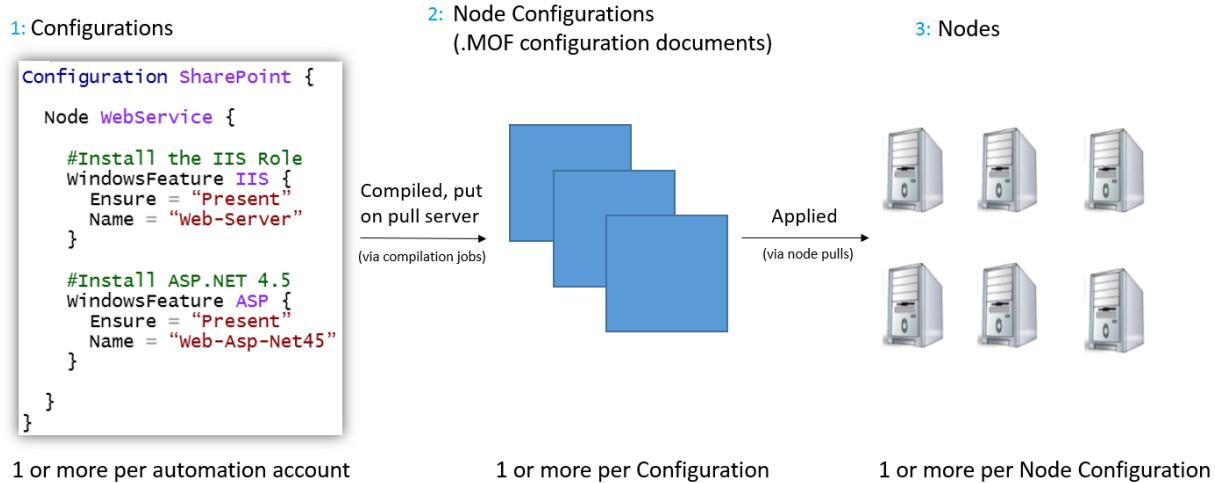
The following outlines some of the reasons why we would consider using Azure Automation DSC:

- Built-in pull server. Provides a DSC pull server like the Windows Feature DSC service so that target nodes automatically receive configurations, conform to the desired state, and report back on their compliance. The built-in pull server in Azure Automation eliminates the need to set up and maintain your pull server.
- Management of all your DSC artifacts. You can manage all your DSC configurations, resources, and target nodes from the Azure portal or PowerShell.
- Import reporting data into Log Analytics. Nodes managed with Azure Automation state configuration send detailed reporting status data to the built-in pull server. You can configure Azure Automation state configuration to send this data to your Log Analytics workspace.

## How Azure Automation state configuration works

The general process for how Azure Automation State configuration works is as follows:

1. Create a PowerShell script with the configuration element.
2. Upload the script to Azure Automation and compile the script into a MOF file. The file is transferred to the DSC pull server, provided as part of the Azure Automation service.
3. Define the nodes that will use the configuration, and then apply the configuration.



For more information, visit [Managed Object Format \(MOF\) file](#).

## Examine DSC configuration file

Completed 100 XP

- 2 minutes

**DSC configurations** are Windows PowerShell scripts that define a special type of function.

You can view some syntax examples and scenarios on the [Configuration syntax](#) page.

## DSC configuration elements

We'll provide the example configurations and then discuss the elements within them. Let's start with the following example configuration:

PowerShellCopy

```
configuration LabConfig
{
    Node WebServer
    {
        WindowsFeature IIS
        {
            Ensure = 'Present'
            Name = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }
}
```

- **Configuration block.** The **Configuration** block is the outermost script block. In this case, the name of the configuration is **LabConfig**. Notice the curly brackets to define the block.
- **Node block.** There can be one or more **Node** blocks. It defines the nodes (computers and VMs) that you're configuring. In this example, the node targets a computer called **WebServer**. You could also call it **localhost** and use it locally on any server.
- **Resource blocks.** There can be one or more resource blocks. It's where the configuration sets the properties for the resources. In this case, there's one resource block called **WindowsFeature**. Notice the parameters that are defined. (You can read more about resource blocks at [DSC resources](#).)

Here's another example:

PowerShellCopy

```
Configuration MyDscConfiguration
{
    param
```

```

(
    [string[]]$ComputerName='localhost'
)

Node $ComputerName
{
    WindowsFeature MyFeatureInstance
    {
        Ensure = 'Present'
        Name = 'RSAT'
    }

    WindowsFeature My2ndFeatureInstance
    {
        Ensure = 'Present'
        Name = 'Bitlocker'
    }
}

MyDscConfiguration

```

In this example, you specify the node's name by passing it as the *ComputerName* parameter when you compile the configuration. The name defaults to "localhost."

Within a Configuration block, you can do almost anything that you normally could in a PowerShell function.

You can also create the configuration in any editor, such as PowerShell ISE, and save the file as a PowerShell script with a .ps1 file type extension.

## Exercise - Import and compile

Completed 100 XP

- 2 minutes

After creating your DSC configuration file, you must import it and compile it to the DSC pull server. Compiling will create the MOF file. Read more about it at [Compiling a DSC Configuration with the Azure portal](#).

## Import and compile configurations

To import and compile a configuration, complete the following high-level steps:

1. Create a configuration file by creating a file on your local machine. Then, copy and paste the following PowerShell code into the file, and name it **LabConfig.ps1**. This script configuration will ensure the IIS web-server role is installed on the servers:

PowerShellCopy

```
configuration LabConfig
{
    Node WebServer
    {
        WindowsFeature IIS
        {
            Ensure = 'Present'
            Name = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }
}
```

2. In Azure Automation, account under **Configuration Management > State configuration (DSC)**, select the **Configurations** tab, and select **+Add**.

The screenshot shows the 'az-auto-ac-1 - State configuration (DSC)' blade. On the left, there's a navigation menu with 'Configuration Management' (selected), 'Inventory', 'Change tracking', and 'State configuration (DSC)' (selected). At the top, there are buttons for '+ Add', 'Compose configuration', 'Refresh', and 'Reset filters'. Below these are tabs for 'Nodes', 'Configurations' (selected), 'Compiled configurations', and 'Gallery'. A search bar says 'Search configurations...'. The main area has columns for 'CONFIGURATION', 'COMPILED CONFIGURATION COUNT', and 'LAST MODIFIED'. It displays 'No data'.

3. Point to the configuration file you want to import, and then select **OK**.



## Import

Configuration



Add a new configuration or update an existing one. Select a file smaller than 1 MB to import.

\* Configuration file ⓘ

"LabConfig.ps1"



\* Name

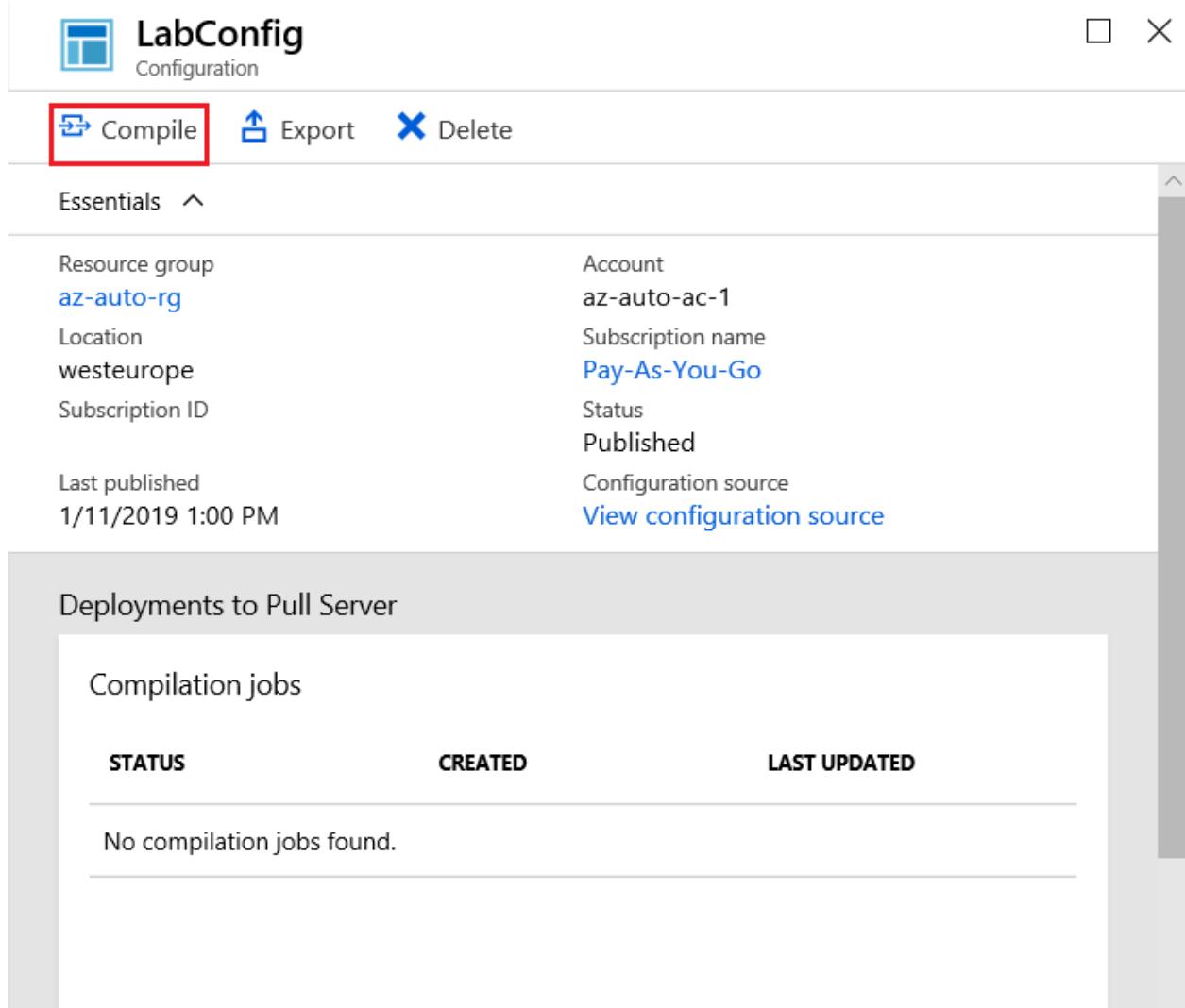
LabConfig

Description



OK

4. Once imported, double click the file, select **Compile**, and then confirm by selecting **Yes**.

A screenshot of the LabConfig Configuration interface. The title bar shows "LabConfig Configuration". Below the title bar are three buttons: "Compile" (highlighted with a red box), "Export", and "Delete". The main content area is titled "Essentials ^". It displays configuration details in two columns: "Resource group" (az-auto-rg), "Location" (westeurope), "Subscription ID", "Account" (az-auto-ac-1), "Subscription name" (Pay-As-You-Go), "Status" (Published), and "Last published" (1/11/2019 1:00 PM). A link "View configuration source" is also present. Below this, a section titled "Deployments to Pull Server" contains a table for "Compilation jobs". The table has columns: STATUS, CREATED, and LAST UPDATED. A message "No compilation jobs found." is displayed.

| STATUS                     | CREATED | LAST UPDATED |
|----------------------------|---------|--------------|
| No compilation jobs found. |         |              |

5. Once compiled, verify that the file has a status of completed.

The screenshot shows the LabConfig application interface. At the top, there's a header with the title "LabConfig" and a "Configuration" sub-header. Below the header are three buttons: "Compile" (with a gear icon), "Export" (with a file icon), and "Delete" (with a trash bin icon). A "Essentials" section is expanded, showing configuration details:

|                            |   |
|----------------------------|---|
| Resource group             | Account                                   |
| <a href="#">az-auto-rg</a> | <a href="#">az-auto-ac-1</a>              |
| Location                   | Subscription name                         |
| westeurope                 | <a href="#">Pay-As-You-Go</a>             |
| Subscription ID            | Status                                    |
|                            | Published                                 |
| Last published             | Configuration source                      |
| 1/11/2019 1:00 PM          | <a href="#">View configuration source</a> |

Below this, a section titled "Deployments to Pull Server" contains a table for "Compilation jobs".

| STATUS   | CREATED           | LAST UPDATED      |
|--|-------------------|-------------------|
| <span style="color: green;">✓ Completed</span> | 1/11/2019 1:05 PM | 1/11/2019 1:05 PM |

### Note

If you prefer, you can also use the **PowerShell Start-AzAutomationDscCompilationJob** cmdlet. More information about this method is available at [Compiling a DSC Configuration with Windows PowerShell](#).

## Exercise - Onboard machines for management

Completed 100 XP

- 3 minutes

After your configuration is in place, you'll select the Azure VMs or on-premises VMs that you want to onboard.

## Note

For more information on onboardin on-premises VMs, review the [Physical/virtual Windows machines on-premises or in a cloud other than the Azure/AWS](#) webpage.

You can onboard a VM and enable DSC in several different ways. Here we'll cover onboardin through an Azure Automation account.

## Onboard VMs to configure

When onboardin VMs using this method, you'll need to deploy your VMs to Azure before starting:

1. In the left pane of the Automation account, select **State configuration (DSC)**.
2. Select the **Nodes** tab, and then select **+ Add** to open the Virtual Machines pane.
3. Find the VM you would like to enable. (You can use the search field and filter options to find a specific VM, if necessary.)
4. Select the VM, and then select **Connect**.

The screenshot shows the Azure portal interface. On the left, there's a sidebar with 'Virtual Machines' and 'az-auto-ac-1'. The main area has a 'Virtual Machines' heading and a 'SimpleWinVM' card. The card includes a 'Connect' button (which is highlighted with a red box), a 'Refresh' button, and a 'Learn more' link. Below the card, a status bar shows 'Not connected', 'POWER STATE VM running', 'OS Windows', and 'STATUS Not connected'. On the right, a detailed table lists two virtual machines:

| VIRTUAL MACHINES | SUBSCRIPTION  | RESOURCE GROUP | LOCATION    |
|------------------|---------------|----------------|-------------|
| SimpleWinVM      | Pay-As-You-Go | simplewinvm    | West Europe |
| SimpleWinVM      | Pay-As-You-Go | simplewivm_rg2 | West Europe |

5. In the resultant Registration pane, configure the following settings, and then select **OK**.



## Registration

□ X

\* Registration key

Primary key Secondary key

Node configuration name ⓘ

LabConfig.WebServer



Refresh Frequency ⓘ

30

Configuration Mode Frequency ⓘ

15

Configuration Mode ⓘ

ApplyAndMonitor



Allow Module Override ⓘ

Reboot Node if Needed ⓘ

Action after Reboot ⓘ

ContinueConfiguration



### Property

#### Description

Registration key.

Primary or secondary, for registering the node with a pull service.

Node configuration name.

The name of the node configuration that the VM should be configured to pull for Automation DSC.

Refresh Frequency.

The time interval, in minutes, at which the LCM checks a pull service to get updated configurations. This value is ignored if the LCM isn't configured in pull mode. The default value is 30.

Configuration Mode Frequency.

How often, in minutes, the current configuration is checked and applied. This property is ignored if the **ConfigurationMode** property is set to **ApplyOnly**. The default value is 15.

Configuration mode.

Specifies how the LCM gets configurations. Possible values are **ApplyOnly**, **ApplyAndMonitor**, and **ApplyAndAutoCorrect**.

Allow Module Override.

Controls whether new configurations downloaded from the Azure Automation DSC pull server can overwrite the old modules already on the target server.

Reboot Node if Needed.

Set this to **\$true** to automatically reboot the node after a configuration that requires a reboot is applied. Otherwise, you'll have to reboot the node for any configuration that needs it manually. The default value is **\$false**.

Action after Reboot.

Specifies what happens after a reboot during the application of a configuration. The possible values are **ContinueConfiguration** and **StopConfiguration**.

The service will then connect to the Azure VMs and apply the configuration.

6. Return to the State configuration (DSC) pane and verify that the status now displays as Compliant after applying the configuration.

[Add](#)  [Refresh](#)  [Reset filters](#)  [Enable Log Search](#)

[Nodes](#) [Configurations](#) [Compiled configurations](#) [Gallery](#)

**Configuration status**

| Status        | Count |
|---------------|-------|
| Compliant     | 1     |
| Pending       | 0     |
| In progress   | 0     |
| Not compliant | 0     |
| Failed        | 0     |
| Unresponsive  | 0     |

**Nodes**  [ⓘ](#) **Status**  [ⓘ](#) **Node configuration**  [ⓘ](#) **VM DSC extension version > 2.70**  [ⓘ](#)

Search Node names... 6 selected All 2 selected

| NODE        | STATUS    | NODE CONFIGURATION  | LAST SEEN         | VERSION  | ... |
|-------------|-----------|---------------------|-------------------|----------|-----|
| SimpleWinVM | Compliant | LabConfig.WebServer | 1/11/2019 2:17 PM | 2.77.0.0 | ... |

Each time Azure Automation DSC does a consistency check on a managed node, the node sends a status report back to the pull server. You can review these reports on that node's blade. Access it by double-clicking or pressing the spacebar and then Enter on the node.

The screenshot shows the Azure portal interface for a node named "SimpleWinVM". At the top, there's a blue icon followed by the node name "SimpleWinVM" and three standard window control buttons (minimize, maximize, close). Below the title, there are two buttons: "Assign node configuration" with a gear icon and "Unregister" with a cross icon. A section titled "Essentials" is expanded, showing various configuration details:

|                                      |                     |
|--------------------------------------|---------------------|
| Resource group                       | IP address          |
| az-auto-rg                           | 10.0.0.4            |
| Id                                   | Account             |
| b6bc9bd8-15a8-11e9-a80e-000d3a4664d5 | az-auto-ac-1        |
| Last seen time                       | Virtual machine     |
| 1/11/2019 2:27 PM                    | SimpleWinVM         |
| Configuration                        | Node configuration  |
| LabConfig                            | LabConfig.WebServer |
| Registration time                    | Status              |
| 1/11/2019 2:09 PM                    | Compliant           |

Below this, there's a section titled "Reports" which lists four entries:

| TYPE        | STATUS      | REPORT TIME       |
|-------------|-------------|-------------------|
| Consistency | ✓ Compliant | 1/11/2019 2:27 PM |
| Consistency | ✓ Compliant | 1/11/2019 2:27 PM |
| Initial     | ✓ Compliant | 1/11/2019 2:07 PM |
| Consistency | ✓ Compliant | 1/11/2019 2:12 PM |

## Note

You can also unregister the node and assign a different configuration to nodes.

For more information about onboarding VMs, see also:

- [Enable Azure Automation State Configuration.](#)
- [Configuring the Local Configuration Manager](#)

## Explore hybrid management

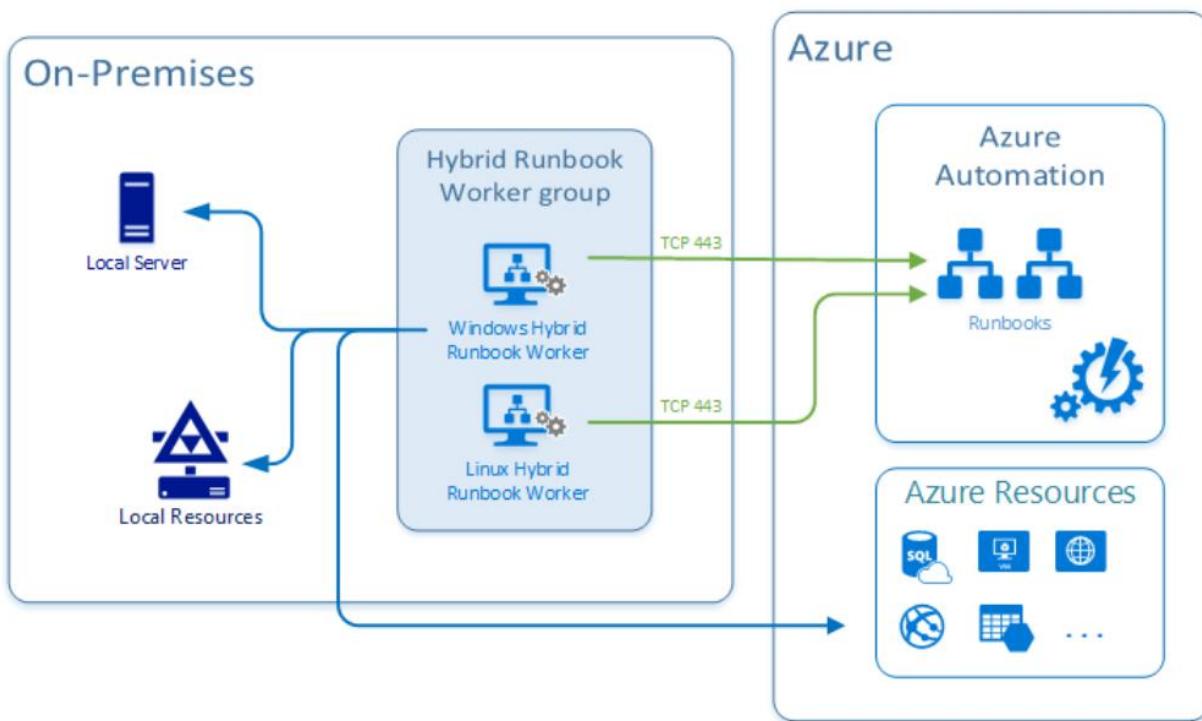
Completed 100 XP

- 2 minutes

The Hybrid Runbook Worker feature of Azure Automation allows you to run runbooks that manage local resources in your private data center on machines located in your data center.

Azure Automation stores and manages the runbooks and then delivers them to one or more on-premises machines.

The Hybrid Runbook Worker functionality is presented in the following graphic:



## Hybrid Runbook Worker workflow and characteristics

The following list is characteristics of the Hybrid Runbook Worker workflow:

- You can select one or more computers in your data center to act as a Hybrid Runbook Worker and then run runbooks from Azure Automation.
- Each Hybrid Runbook Worker is a member of a Hybrid Runbook Worker group, which you specify when you install the agent.
- A group can include a single agent, but you can install multiple agents in a group for high availability.

- There are no inbound firewall requirements to support Hybrid Runbook Workers, only Transmission Control Protocol (TCP) 443 is required for outbound internet access.
- The agent on the local computer starts all communication with Azure Automation in the cloud.
- When a runbook is started, Azure Automation creates an instruction that the agent retrieves. The agent then pulls down the runbook and any parameters before running it.

To configure your on-premises servers that support the Hybrid Runbook Worker role with DSC, you must add them as DSC nodes.

For more information about onboarding them for management with DSC, see [Onboarding machines for management by Azure Automation State Configuration](#).

For more information on installing and removing Hybrid Runbook Workers and groups, see:

- [Automate resources in your datacenter or cloud by using Hybrid Runbook Worker](#).
- [Hybrid Management in Azure Automation](#)

## Implement DSC and Linux Automation on Azure

Completed 100 XP

- 1 minute

Follow more specific, and up-to-date details are available at [Get started with Desired State Configuration \(DSC\) for Linux](#).

The following Linux operating system versions are currently supported by both PowerShell DSC and Azure Automation DSC:

- CentOS 6, 7, and 8 (x64)
- Debian GNU/Linux 8, 9, and 10 (x64)
- Oracle Linux 6 and 7 (x64)
- Red Hat Enterprise Linux Server 6, 7, and 8 (x64)
- SUSE Linux Enterprise Server 12 and 15 (x64)
- Ubuntu Server 14.04 LTS, 16.04 LTS, 18.04 LTS, and 20.04 LTS (x64)

# Explore Azure Automation with DevOps

## Create automation accounts

Completed 100 XP

- 4 minutes

To start using the Microsoft Azure Automation service, you must create an [Automation account](#) from within the Azure portal.

Steps to create an Azure Automation account are available on the [Create an Azure Automation account](#) page.

Automation accounts are like Azure Storage accounts, serving as a container to store automation artifacts.

These artifacts could be a container for all your runbooks, runbook executions (*jobs*), and the assets on which your runbooks depend.

An Automation account allows you to manage all Azure resources via an API. To safeguard it, the Automation account creation requires subscription-owner access.

Home > New > Add Automation Account

## Add Automation Account

\* Name i  
azautoac01 ✓

\* Subscription  
Pay-As-You-Go

\* Resource group  
(New) az-auto-rg ✓

[Create new](#)

\* Location  
Japan East

\* Create Azure Run As account i  
Yes No

i The Run As account feature will create a Run As account and a Classic Run As account. [Click here to learn more about Run As accounts.](#)

i Learn more about Automation pricing.

Create

You must be a subscription owner to create the Run As accounts that the service creates.

If you don't have the proper subscription privileges, you'll see the following warning:



You do not have permissions to create an Azure Run As account (service principal) and grant Contributor role to the service principal. Please follow the directions in this document to create one with the help of the subscription admin. [Click here to learn more about Run As accounts.](#)

You'll need at least one Azure Automation account to use Azure Automation.

However, as a best practice, you should create multiple automation accounts to segregate and limit the scope of access and minimize any risk to your organization.

For example, you might use one account for development, another for production, and another for your on-premises environment. You can have up to 30 Automation accounts.

## What is a runbook?

Completed 100 XP

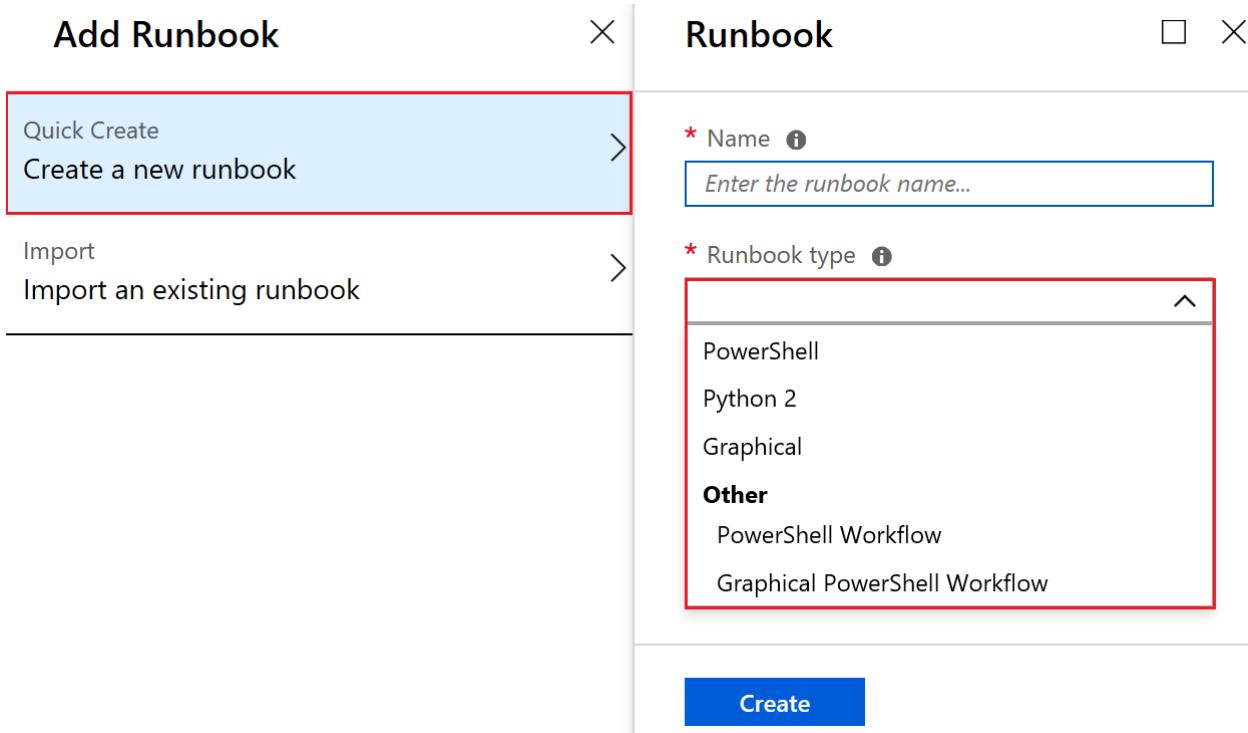
- 4 minutes

**Runbooks** serve as repositories for your custom scripts and workflows.

They also typically reference Automation shared resources such as credentials, variables, connections, and certificates.

Runbooks can also contain other runbooks, allowing you to build more complex workflows.

You can invoke and run runbooks on-demand or according to a schedule using Automation Schedule assets.



## Creating runbooks

When creating runbooks, you have two options. You can either:

- Create your runbook and import it. For more information about creating or importing a runbook in Azure Automation, go to [Start a runbook in Azure Automation](#).
- Modify runbooks from the runbook gallery. It provides a rich ecosystem of runbooks that are available for your requirements. Visit [Runbook and module galleries for Azure Automation](#) for more information.

A vibrant open-source community also creates runbooks you can apply directly to your use cases.

You can choose from different runbook types based on your requirements and Windows PowerShell experience.

If you prefer to work directly with Windows PowerShell code, you can use a PowerShell runbook or a PowerShell Workflow runbook.

You can edit offline or with the textual editor in the Azure portal using either of these.

If you prefer to edit a runbook without exposure to the underlying code, you can create a graphical runbook using the Azure portal's graphic editor.

## Graphical runbooks

Graphical runbooks and Graphical PowerShell Workflow runbooks are created and edited with the graphic editor in the Azure portal.

You can export them to a file and import them into another automation account, but you can't create or edit them with another tool.

## PowerShell runbooks

PowerShell runbooks are based on Windows PowerShell. You edit the runbook code directly using the text editor in the Azure portal.

You can also use any offline text editor and import the runbook into Azure Automation. PowerShell runbooks don't use parallel processing.

## PowerShell Workflow runbooks

PowerShell Workflow runbooks are text runbooks based on Windows PowerShell Workflow.

You directly edit the runbook code using the text editor in the Azure portal.

You can also use any offline text editor and import the runbook into Azure Automation.

PowerShell Workflow runbooks use parallel processing to allow for the simultaneous completion of multiple tasks.

Workflow runbooks take longer to start than PowerShell runbooks because they must be compiled before running.

## Python runbooks

You can directly edit the code of the runbook using the text editor in the Azure portal, or you can use any offline text editor and import the runbook into Azure Automation.

You can also use Python libraries. You must first import the package into the Automation Account to use third-party libraries.

### Note

You can't convert runbooks from graphical to textual type and vice versa.

For more information on the different types of runbooks, visit [Azure Automation runbook types](#).

## Understand automation shared resources

Completed 100 XP

- 3 minutes

Azure Automation contains shared resources that are globally available to be associated with or used in a runbook.

There are currently eight shared resources categories:

- **Schedules**: It allows you to define a one-off or recurring schedule.
- **Modules**: Contains Azure PowerShell modules.
- **Modules gallery**: It allows you to identify and import PowerShell modules into your Azure automation account.
- **Python packages**: Allows you to import a Python package by uploading: **.whl** or **tar.gz** packages.
- **Credentials**: It allows you to create username and password credentials.
- **Connections**: It allows you to specify Azure, Azure classic certificate, or Azure Service principal connections.
- **Certificates**: It allows you to upload certificates in **.cer** or **pfx** format.
- **Variables**: It allows you to define encrypted or unencrypted variables of types—for example, *String*, *Boolean*, *DateTime*, *Integer*, or no specific type.

## Shared Resources

---

-  Schedules
-  Modules
-  Python packages
-  Credentials
-  Connections
-  Certificates
-  Variables

As a best practice, always try to create global assets to be used across your runbooks.

It will save time and reduce the number of manual edits within individual runbooks.

## Explore runbook gallery

Completed 100 XP

- 3 minutes

Azure Automation runbooks are provided to help eliminate the time it takes to build custom solutions.

The runbooks have already been built by Microsoft and the Microsoft community.

You can use them with or without modification.

Also, you can import runbooks from the runbook gallery at Azure Automation Github in the runbooks repository [Azure Automation - Runbooks](#).

### Note

A new Azure PowerShell module was released in December 2018, called the **Az** PowerShell module. It replaces the **AzureRM** PowerShell module and is now the intended PowerShell module for interacting with Azure. This new **Az** module is currently supported in Azure Automation. For more general details on the new Az PowerShell module, go to [Introducing the new Azure PowerShell Az module](#).

## Choosing items from the runbook gallery

In the Azure portal, you can import directly from the runbook gallery using the following high-level steps:

1. Open your Automation account, and then select **Process Automation > Runbooks**.
2. In the runbooks pane, select **Browse gallery**.
3. From the runbook gallery, locate the runbook item you want, select it, and select **Import**.

When browsing through the runbooks in the repository, you can review the code or visualize the code.

You can also check information such as the source project and a detailed description, ratings, and questions and answers.

For more information, see [Azure Automation](#).

Start Azure V2 VMs

[View Source](#)

**Import**

This Graphical PowerShell runbook connects to Azure using an Automation Run As account and starts all V2 VMs in an Azure subscription or in a resource group or a single named V2 VM. You can attach a recurring schedule to this runbook to run it at a specific time. The asso

Created by: Azure Automation Product Team - **Microsoft**

Tags: [Azure Virtual Machines](#), [Start VM](#), [GraphicalPS](#)

[View Source Project](#)

Ratings: 4.43 of 5  
75,721 downloads  
Last updated: 10/22/2016

```

graph TD
    A[Get Run As Connection] --> B[Connect to Azure]
    B --> C[Get single VM]
    B --> D[Get all VMs in RG]
    B --> E[Get all VMs in Sub]
    F[READ ME]
  
```

**ATTENTION**

Each runbook is licensed to you under a license agreement by its owner, not Microsoft. Microsoft is not responsible for runbooks provided & licensed by the community members and does not screen for security, compatibility or performance. The runbooks are not supported under any Microsoft support program or service. The runbooks are provided AS IS without warranty of any kind.

## Note

Python runbooks are also available from the Azure Automation Github in the runbooks repository. To find them, filter by language and select **Python**.

## Note

You can't use PowerShell to import directly from the runbook gallery.

# Examine webhooks

Completed 100 XP

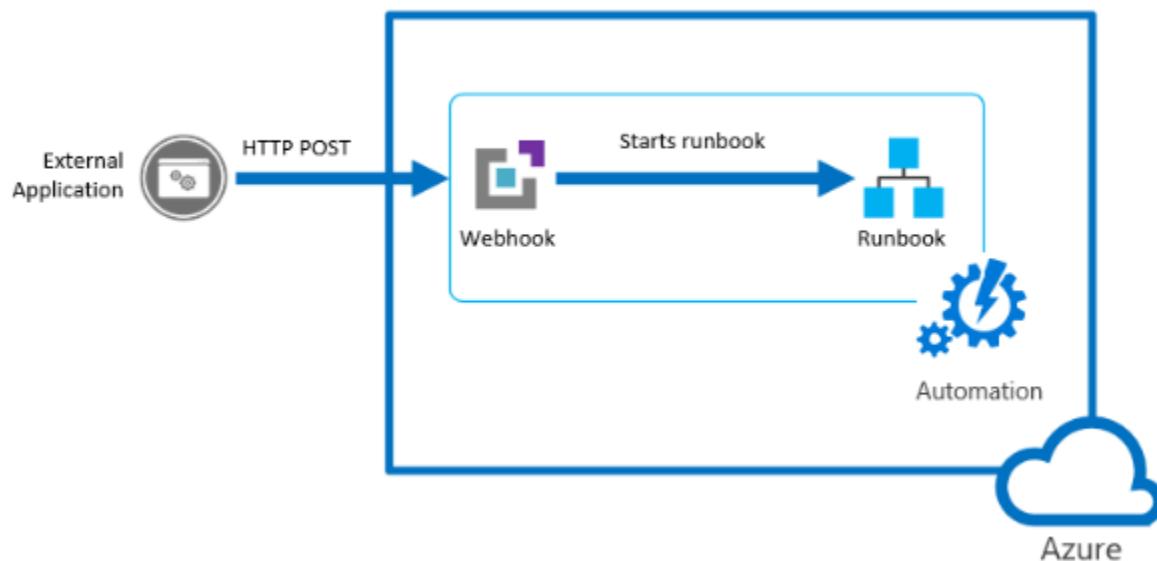
- 4 minutes

You can automate starting a runbook either by scheduling it or by using a webhook.

A **webhook** allows you to start a particular runbook in Azure Automation through a single HTTPS request.

It allows external services such as Azure DevOps, GitHub, or custom applications to start runbooks without implementing more complex solutions using the Azure Automation API.

More information about webhooks is available at [Starting an Azure Automation runbook with a webhook](#).

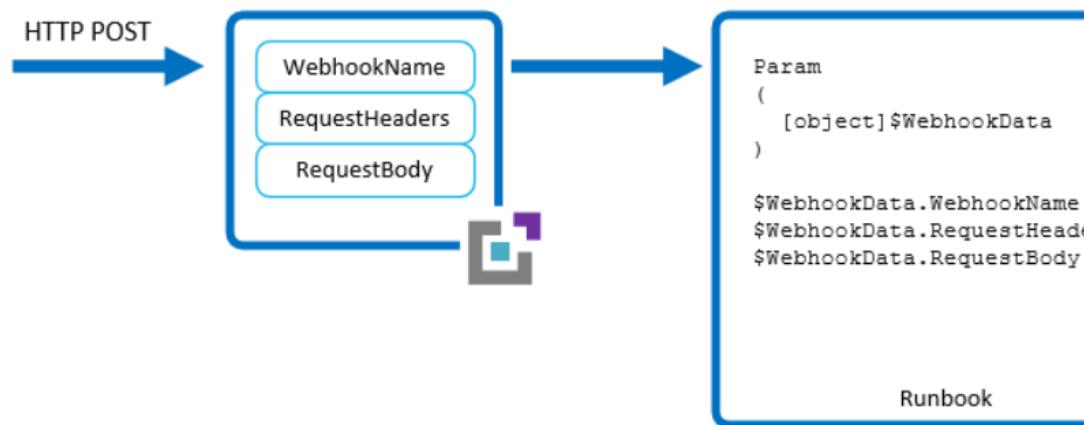


## Create a webhook

You create a webhook linked to a runbook using the following steps:

1. In the Azure portal, open the runbook that you want to create the webhook.
2. In the runbook pane, under Resources, select **Webhooks**, and then choose **+ Add webhook**.
3. Select **Create new webhook**.
4. In the **Create new webhook** dialog, there are several values you need to configure. After you configure them, select **Create**:
  - **Name**. Specify any name you want for a webhook because the name isn't exposed to the client. It's only used for you to identify the runbook in Azure Automation.
  - **Enabled**. A webhook is enabled by default when it's created. If you set it to Disabled, then no client can use it.

- **Expires.** Each webhook has an expiration date, at which time it can no longer be used. You can continue to modify the date after creating the webhook providing the webhook isn't expired.
- **URL.** The webhook URL is the unique address that a client calls with an HTTP POST to start the runbook linked to the webhook. It's automatically generated when you create the webhook, and you can't specify a custom URL. The URL contains a security token that allows the runbook to be invoked by a third-party system with no further authentication. For this reason, treat it like a password. You can only view the URL in the Azure portal for security reasons when the webhook is created. Make a note of the URL in a secure location for future use.



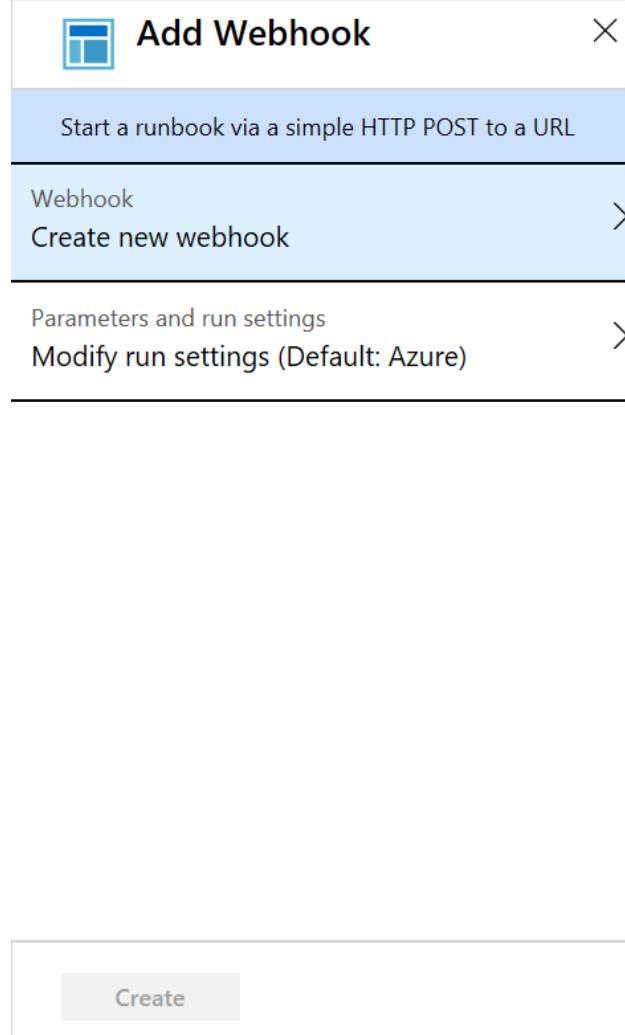
## Note

When creating it, make sure you copy the webhook URL and then store it in a safe place. After you create the webhook, you can't retrieve the URL again.

5. Select the **Parameters run settings (Default: Azure)** option. This option has the following characteristics, which allows you to complete the following actions:

- If the runbook has mandatory parameters, you'll need to provide these required parameters during creation. You aren't able to create the webhook unless values are provided.
- If there are no mandatory parameters in the runbook, there's no configuration required here.
- The webhook must include values for any mandatory parameters of the runbook and include values for optional parameters.
- When a client starts a runbook using a webhook, it can't override the parameter values defined.

- To receive data from the client, the runbook can accept a single parameter called `$WebhookData` of type [object] that contains data that the client includes in the POST request.
- There's no required webhook configuration to support the `$WebhookData` parameter.



The screenshot shows the 'Add Webhook' dialog box. On the left, there's a sidebar with a 'Webhook' section containing a 'Create new webhook' button. Below it is a 'Parameters and run settings' section with a 'Modify run settings (Default: Azure)' button. On the right, the main area is titled 'Create a new webhook'. It contains a warning icon with the text: 'For security, after creating a webhook its URL can't be viewed. Make sure to copy it before pressing "OK", and to store it securely.' Below this are fields for 'Name' (with placeholder 'Enter the webhook name...'), 'Enabled' (with 'Yes' and 'No' buttons), 'Expires' (set to '2020-01-11' at '6:00:23 AM'), and a 'URL' field containing 'https://s2events.azure-automation.net'. At the bottom are 'Create' and 'OK' buttons.

6. When finished, select **Create**.

## Using a webhook

To use a webhook after it has been created, your client application must issue an HTTP POST with the URL for the webhook.

- The syntax of the webhook is in the following format:

[Copy](#)

`http://< Webhook Server >/token?=< Token Value >`

- The client receives one of the following return codes from the POST request.

Expand table

| Code Test                 | Description  |
|---------------------------|--|
| 202 Accepted              | The request was accepted, and the runbook was successfully queued.                                   |
| 400 Bad request           | The request wasn't accepted because the runbook has expired, been disabled, or is currently running. |
| 404 Not found             | The request wasn't accepted because the webhook, runbook, or account wasn't found.                   |
| 500 Internal Server Error | An unexpected error occurred on the server.  |

- If successful, the webhook response contains the job ID in JSON format as follows:

JSONCopy

```
{"JobIds": ["< JobId >"]}
```

The response will contain a single job ID, but the JSON format allows for potential future enhancements.

- You can't determine when the runbook job completes or determine its completion status from the webhook. You can only choose this information using the job ID with another method such as PowerShell or the Azure Automation API.

More details are available on the [Starting an Azure Automation runbook with a webhook](#) page.

## Explore source control integration

Completed 100 XP

- 4 minutes

Azure Automation supports source control integration that enables you to keep your runbooks in your Automation account up to date with your scripts in your GitHub or Azure DevOps source control repository.

Source control allows you to collaborate with your team more efficiently, track changes, and roll back to earlier versions of your runbooks.

For example, source control will enable you to sync different branches in source control to your development, test, or production Automation accounts.

It makes it easier to promote code you've tested in your development environment to your production Automation account.

Azure Automation supports three types of source control:

- GitHub.
- Azure DevOps (Git).
- Azure DevOps (TFVC).

Source control allows you to push code from Azure Automation to source control or pull your runbooks from source control to Azure Automation.

Source control sync jobs run under the user's Automation Account and are billed at the same rate as other Automation jobs.

## Integrate source control with Azure Automation

You integrate source control with Azure Automation using the following steps:

1. In the Azure portal, access your Automation account.
2. Under Account Settings, select **Source control**, and then choose **+ Add**.
3. In the **Source Control Summary** blade, select **GitHub** as source control type and then select **Authenticate**.

### Note

Note: You'll require a GitHub account to complete the next step.

4. When the browser page opens, prompting you to authenticate to <https://www.github.com>, select **Authorize azureautomation** and enter your GitHub account password. If successful, you should receive an email

notification from GitHub stating that *A third-party OAuth Application (Automation Source Control) with repo scope was recently authorized to access your account.*

5. After authentication completes, fill in the details based on the following table, and then select **Save**.

Expand table

| Property            | Description  |
|---------------------|--|
| Name                | Friendly name  |
| Source control type | GitHub, Azure DevOps Git, or Azure DevOps TFVC   |
| Repository          | The name of the repository or project  |
| Branch              | The branch from which to pull the source files. Branch targeting isn't available for the GitHub source control type. |
| Folder Path         | The folder that contains the runbooks to sync.   |
| Autosync            | Turns on or off automatic sync when a commit is made in the source control repository.                               |
| Publish Runbook.    | If set to <b>On</b> , after runbooks are synced from source control, they'll be automatically published.             |
| Description         | A text field to provide more details.  |

6. If you set **Autosync** to **Yes**, full sync will start. If you set **Autosync** to **No**, open the **Source Control Summary** blade again by selecting your repository in Azure Automation and then selecting **Start Sync**.

**Source Control Summary**

Start Sync Save Delete Discard

---

\* Source control name  
gituh repo

\* Source control type  
GitHub

Authorization successful.

\* Repository  
PartsUnlimited

\* Branch  
master

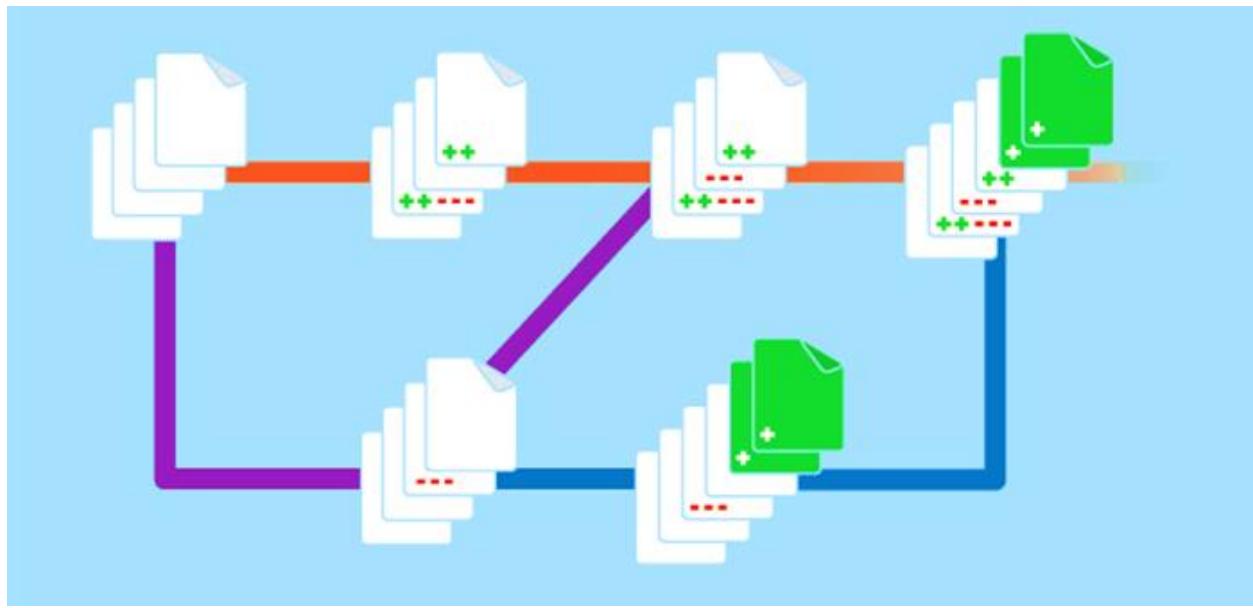
\* Folder path  
\Labfiles\Devops200.2x-InfrastructureasCode\Mod01

Auto Sync   
 On Off

Publish Runbook   
 On Off

Description

7. Verify that your source control is listed on the **Azure Automation Source control** page for you to use.



## Explore PowerShell workflows

Completed 100 XP

- 4 minutes

IT pros often automate management tasks for their multi-device environments by running sequences of long-running tasks or workflows.

These tasks can affect multiple managed computers or devices at the same time.

PowerShell Workflow lets IT pros and developers apply the benefits of Windows Workflow Foundation with the automation capabilities and ease of using Windows PowerShell.

### Tip

Refer to [A Developer's Introduction to Windows Workflow Foundation \(WF\) in .NET 4](#) for more information.

Windows PowerShell Workflow functionality was introduced in Windows Server 2012 and Windows 8 and is part of Windows PowerShell 3.0 and later.

Windows PowerShell Workflow helps automate distribution, orchestration, and completion of multi-device tasks, freeing users and administrators to focus on higher-level tasks.

## Activities

An **activity** is a specific task that you want a workflow to do. Just as a script is composed of one or more commands, a workflow is composed of activities carried out in sequence.

You can also use a script as a single command in another script and use a workflow as an activity within another workflow.

## Workflow characteristics

A workflow can:

- Be long-running.
- Be repeated over and over.
- Run tasks in parallel.
- Be interrupted—can be stopped and restarted, suspended, and resumed.
- Continue after an unexpected interruption, such as a network outage or computer/server restart.

## Workflow benefits

A workflow offers many benefits, including:

- Windows PowerShell scripting syntax. Is built on PowerShell.
- Multidevice management. Simultaneously apply workflow tasks to hundreds of managed nodes.
- Single task runs multiple scripts and commands. Combine related scripts and commands into a single task. Then run the single task on multiple computers. The activity status and progress within the workflow are visible at any time.
- Automated failure recovery.
  - Workflows survive both planned and unplanned interruptions, such as computer restarts.
  - You can suspend a workflow operation, then restart or resume the workflow from the point it was suspended.

- You can author checkpoints as part of your workflow so that you can resume the workflow from the last persisted task (or checkpoint) instead of restarting the workflow from the beginning.
- Connection and activity retries. You can retry connections to managed nodes if network-connection failures occur. Workflow authors can also specify activities that must run again if the activity cannot be completed on one or more managed nodes (for example, if a target computer was offline while the activity was running).
- Connect and disconnect from workflows. Users can connect and disconnect from the computer running the workflow, but the workflow will remain running. For example, suppose you're running the workflow and managing the workflow on two different computers. In that case, you can sign out of or restart the computer from which you're managing the workflow and continue to monitor workflow operations from another computer without interrupting the workflow.
- Task scheduling. You can schedule a task to start when specific conditions are met, as with any other Windows PowerShell cmdlet or script.

## Create a workflow

Completed 100 XP

- 3 minutes

Use a script editor such as the Windows PowerShell Integrated Scripting Environment (ISE) to write the workflow.

It enforces workflow syntax and highlights syntax errors. For more information, review the tutorial [Tutorial - Create a PowerShell Workflow runbook in Azure Automation](#).

A benefit of using PowerShell ISE is that it automatically compiles your code and allows you to save the artifact.

Because the syntactic differences between scripts and workflows are significant, a tool that knows both workflows and scripts will save you considerable coding and testing time.

## Syntax

When you create your workflow, begin with the **workflow** keyword, which identifies a workflow command to PowerShell.

A script workflow requires the **workflow** keyword. Next, name the workflow, and have it follow the **workflow** keyword.

The body of the workflow will be enclosed in braces.

1. A workflow is a Windows command type, so select a name with a verb-noun format:

```
PowerShellCopy  
workflow Test-Workflow  
{  
    ...  
}
```

2. To add parameters to a workflow, use the **Param** keyword. It's the same techniques that you use to add parameters to a function.
3. Finally, add your standard PowerShell commands.

```
PowerShellCopy  
workflow MyFirstRunbook-Workflow  
{  
    Param(  
        [string]$VMName,  
        [string]$ResourceGroupName  
    )  
    ...  
    Start-AzureRmVM -Name $VMName -ResourceGroupName $ResourceGroupName  
}
```

## Explore hybrid management

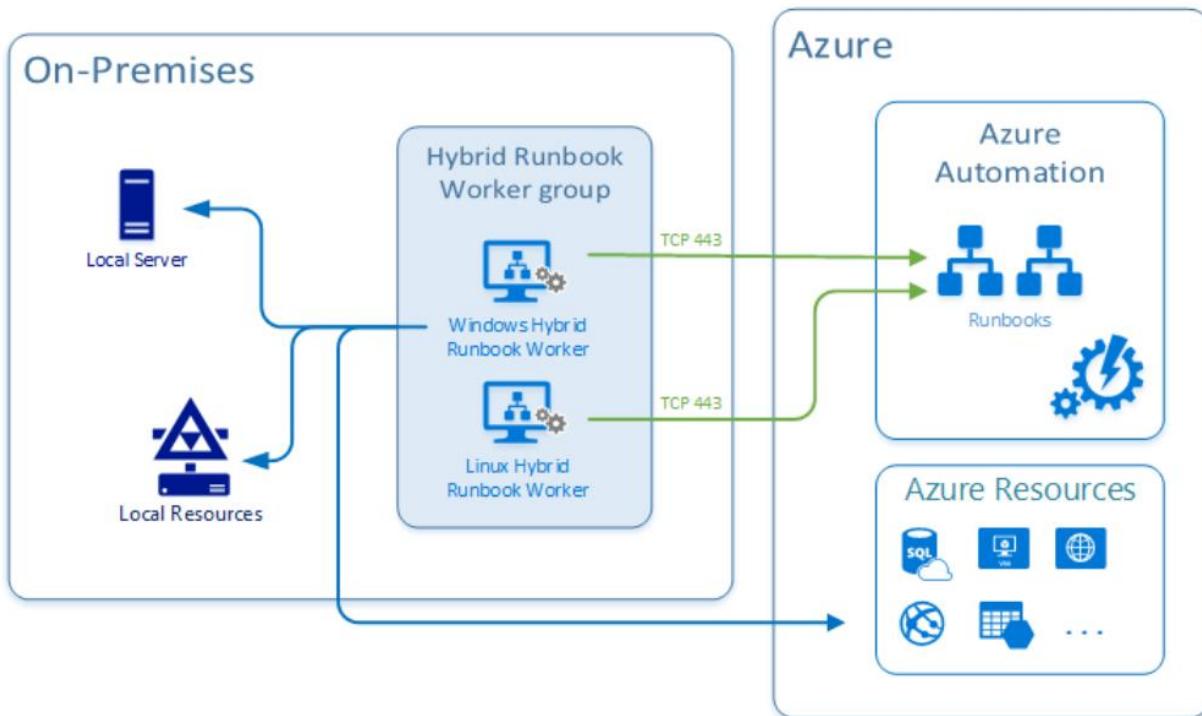
Completed 100 XP

- 3 minutes

The Hybrid Runbook Worker feature of Azure Automation allows you to run runbooks that manage local resources in your private data center on machines located in your data center.

Azure Automation stores and manages the runbooks and then delivers them to one or more on-premises machines.

The Hybrid Runbook Worker functionality is presented in the following graphic:



## Hybrid Runbook Worker workflow and characteristics

The following list is characteristics of the Hybrid Runbook Worker workflow:

- You can select one or more computers in your data center to act as a Hybrid Runbook Worker and then run runbooks from Azure Automation.
- Each Hybrid Runbook Worker is a member of a Hybrid Runbook Worker group, which you specify when you install the agent.
- A group can include a single agent, but you can install multiple agents in a group for high availability.
- There are no inbound firewall requirements to support Hybrid Runbook Workers, only Transmission Control Protocol (TCP) 443 is required for outbound internet access.
- The agent on the local computer starts all communication with Azure Automation in the cloud.
- When a runbook is started, Azure Automation creates an instruction that the agent retrieves. The agent then pulls down the runbook and any parameters before running it.

To configure your on-premises servers that support the Hybrid Runbook Worker role with DSC, you must add them as DSC nodes.

For more information about onboarding them for management with DSC, see [Onboarding machines for management by Azure Automation State Configuration](#).

For more information on installing and removing Hybrid Runbook Workers and groups, see:

- [Automate resources in your datacenter or cloud by using Hybrid Runbook Worker](#).
- [Hybrid Management in Azure Automation](#)

## Exercise - Create and run a workflow runbook

Completed 100 XP

- 30 minutes

### Note

You require an Azure subscription to do the following steps. If you don't have one, you can create one by following the steps outlined on the [Create your Azure free account today](#) webpage.

## Steps

Create a new runbook

1. In the **Azure portal**, open your Automation account.
2. Under **Process Automation**, select **Runbooks** to open the list of runbooks.
3. Create a new runbook by selecting the **Create a new runbook**.
4. Give the runbook the name **MyFirstRunbook-Workflow**.
5. You're going to create a PowerShell Workflow runbook, so for **Runbook type**, select **PowerShell Workflow**.
6. Select **Create** to create the runbook and open the text editor.

Add code to a runbook

You have two options when adding code to a runbook.

You can type code directly into the runbook or select cmdlets, runbooks, and assets from the Library control and have them added to the runbook, along with any related parameters.

For this walkthrough, you'll use the type directly into the runbook method, as detailed in the following steps:

1. Type **Write-Output "Hello World."** between the braces, as per the below:

PowerShellCopy

```
Workflow MyFirstRunbook-Workflow
{
    Write-Output "Hello World"
}
```

2. Save the runbook by selecting **Save**.



## Test the runbook

Before you publish the runbook to production, you want to test it to ensure it works properly.

When you test a runbook, you run the draft version and view its output interactively, as demonstrated in the following steps:

1. Select the **Test** pane.

```
1 workflow MyFirstRunbook-Workflow
2 {
3     Write-Output "Hello World"
4 }
```

2. Select **Start** to start the test. This should be the only enabled option.

Test  
MyFirstRunbook-Workflow

**Start** Stop Suspend Resume View last test Refresh job streams

Parameters  
No input parameters

Run Settings  
Run on Azure

Using a hybrid runbook worker can increase test performance.  
[Learn more](#)

Activity-level tracing  
This configuration is available only for graphical runbooks.

Trace level  
None Basic Detailed

A runbook job is created, and its status is displayed. The job status will start as Queued, indicating that it's waiting for a runbook worker in the cloud to come available. It moves to Starting when a worker claims the job and then Running when the runbook starts running. When the runbook job completes, its output displays. In this case, you should see Hello World.

3. When the runbook job finishes, close the **Test pane**.

The screenshot shows the Azure Runbook Test interface for a runbook named "MyFirstRunbook-Workflow". At the top, there are several buttons: "Start" (highlighted with a dashed blue border), "Stop", "Suspend", "Resume", "View last test", and "Refresh job streams". Below these buttons, the runbook's title "Test" and name "MyFirstRunbook-Workflow" are displayed. The main content area is divided into sections: "Parameters" (No input parameters), "Run Settings" (Run on Azure), and "Activity-level tracing". The "Activity-level tracing" section includes a note about hybrid runbook workers and a "Learn more" link. On the right side, a dark panel displays the status "Completed" and the output "Hello World".

## Publish and run the runbook

The runbook that you created is still in draft mode. It would be best if you published it before you can run it in production. When you publish a runbook, you overwrite the existing published version with the draft version. In your case, you don't have a published version yet because you just created the runbook.

Use the following steps to publish your runbook:

1. In the runbook editor, select **Publish** to publish the runbook.
2. When prompted, select **Yes**.
3. Scroll left to view the runbook in the Runbooks pane and ensure it shows an **Authoring Status** of **Published**.
4. Scroll back to the right to view the pane for **MyFirstRunbook-Workflow**. Notice the options across the top:
  - Start

- View
- Edit
- Link to schedule to start at some time in the future.
- Add a webhook.
- Delete
- Export

MyFirstRunbook-Workflow

Resource group: az-auto-rg

Account: az-auto-ac-1

Location: West Europe

Subscription: Pay-As-You-Go

Subscription ID: 974e6e39-73eb-48b0-9226-dae31425c367

Status: Published

Last modified: 1/11/2019 10:14 AM

Tags: (change) Click here to add tags

| STATUS         | CREATED | LAST UPDATED |
|----------------|---------|--------------|
| No jobs found. |         |              |

5. You want to start the runbook, so select **Start**, and then when prompted, select **Yes**.
6. When the job pane opens for the runbook job you created, leave it open to watch the job's progress.
7. Verify that when the job completes, the job statuses displayed in **Job Summary** match the status you saw when you tested the runbook.



## MyFirstRunbook-Workflow 1/11/2019 10:16 AM

□ X

▶ Resume ■ Stop || Suspend

### Essentials ^

|                                      |   |
|--------------------------------------|---|
| Job Id                               | Created                                 |
| d91f3e85-7196-4fa8-8355-d88c0790ec8c | 1/11/2019 10:16 AM                      |
| Job status                           | Last Update                             |
| Completed                            | 1/11/2019 10:17 AM                      |
| Run As                               | Runbook                                 |
| User                                 | <a href="#">MyFirstRunbook-Workflow</a> |
| Ran on                               | Source snapshot                         |
| Azure                                | <a href="#">View source snapshot</a>    |

### Overview

#### Input

0 ↗

#### Output

➡ Output



All Logs

#### Errors

0 ✘

#### Warnings

0 !

#### Exception

None

# Examine checkpoint and parallel processing

Completed 100 XP

- 4 minutes

Workflows let you implement complex logic within your code. Two features available with workflows are checkpoints and parallel processing.

## Checkpoints

A **checkpoint** is a snapshot of the current state of the workflow.

Checkpoints include the current value for variables and any output generated up to that point. (For more information on what a checkpoint is, read the [checkpoint](#) webpage.)

If a workflow ends in an error or is suspended, the next time it runs, it will start from its last checkpoint instead of at the beginning of the workflow.

You can set a checkpoint in a workflow with the **Checkpoint-Workflow** activity.

For example, if an exception occurs after Activity2, the workflow will end in the following sample code.

When the workflow is rerun, it starts with Activity2, followed just after the last checkpoint set.

PowerShellCopy

```
<Activity1>
  Checkpoint-Workflow
    <Activity2>
      <Exception>
        <Activity3>
```

## Parallel processing

A script block has multiple commands that run concurrently (or *in parallel*) instead of sequentially, as for a typical script.

It's referred to as *parallel processing*. (More information about parallel processing is available on the [Parallel processing](#) webpage.)

In the following example, two *vm0* and *vm1* VMs will be started concurrently, and *vm2* will only start after *vm0* and *vm1* have started.

PowerShellCopy

```
Parallel
{
    Start-AzureRmVM -Name $vm0 -ResourceGroupName $rg
    Start-AzureRmVM -Name $vm1 -ResourceGroupName $rg
}

Start-AzureRmVM -Name $vm2 -ResourceGroupName $rg
```

Another parallel processing example would be the following constructs that introduce some extra options:

- **ForEach -Parallel**. You can use the **ForEach -Parallel** construct to concurrently process commands for each item in a collection. The items in the collection are processed in parallel while the commands in the script block run sequentially.

In the following example, *Activity1* starts at the same time for all items in the collection.

For each item, *Activity2* starts after *Activity1* completes. *Activity3* starts only after both *Activity1* and *Activity2* have been completed for all items.

- **ThrottleLimit** - We use the **ThrottleLimit** parameter to limit parallelism. Too high of a **ThrottleLimit** can cause problems. The ideal value for the **ThrottleLimit** parameter depends on several environmental factors. Try starting with a low **ThrottleLimit** value, and then increase the value until you find one that works for your specific circumstances:

PowerShellCopy

```
ForEach -Parallel -ThrottleLimit 10 ($<item> in $<collection>)
{
    <Activity1>
    <Activity2>
}
<Activity3>
```

A real-world example of it could be similar to the following code: a message displays for each file after it's copied. Only after all files are copied does the completion message display.

PowerShellCopy

```
Workflow Copy-Files
{
    $files =
@("C:\LocalPath\File1.txt","C:\LocalPath\File2.txt","C:\LocalPath\File3.txt")
```

```
ForEach -Parallel -ThrottleLimit 10 ($File in $Files)
{
    Copy-Item -Path $File -Destination \\NetworkPath
    Write-Output "$File copied."
}

Write-Output "All files copied."
```

# Create Azure resources by using Azure CLI

## What is Azure CLI?

Completed 100 XP

- 1 minute

**Azure CLI** is a command-line program you use to connect to Azure and execute administrative commands on Azure resources.

It runs on Linux, macOS, and Windows operating systems.

Instead of a web browser, it allows administrators and developers to execute their commands through a terminal or a command-line prompt (or script).

For example, to restart a VM, you would use a command such as:

Azure CLICopy

```
az vm restart -g MyResourceGroup -n MyVm
```

Azure CLI provides cross-platform command-line tools for managing Azure resources.

You can install it locally on computers running the Linux, macOS, or Windows operating systems.

You can also use Azure CLI from a browser through Azure Cloud Shell.

In both cases, you can use Azure CLI interactively or through scripts:

- Interactive. For Windows operating systems, launch a shell such as cmd.exe, or for Linux or macOS, use Bash. Then issue the command at the shell prompt.
- Scripted. Assemble the Azure CLI commands into a shell script using the script syntax of your chosen shell, and then execute the script.

## Work with Azure CLI

Completed 100 XP

- 3 minutes

Azure CLI lets you control nearly every aspect of every Azure resource.

You can work with Azure resources such as resource groups, storage, VMs, Microsoft Entra ID, containers, and machine learning.

Commands in the CLI are structured in **groups** and **subgroups**.

Each group represents a service provided by Azure, and the subgroups divide commands for these services into logical groupings.

So, how do you find the commands you need? One way is to use the **az find** command.

For example, if you want to find commands that might help you manage a storage blob, you can use the following **find** command:

Azure CLICopy

```
az find blob
```

If you know the command's name you want, the help argument for that command will get you more detailed information on the command—also, a list of the available subcommands for a command group.

For example, here's how you would get a list of the subgroups and commands for managing blob storage:

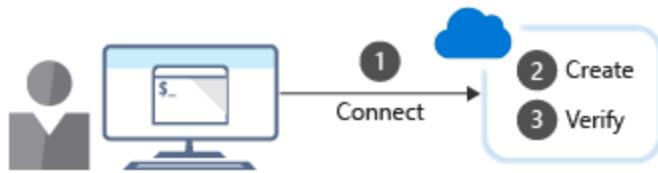
Azure CLICopy

```
az storage blob --help
```

## Creating resources

When creating a new Azure resource, typically, there are three high-level steps:

1. Connect to your Azure subscription.
2. Create the resource.
3. Verify that creation was successful.



## 1. Connect

Because you're working with a local Azure CLI installation, you'll need to authenticate before you can execute Azure commands.

You do it by using the Azure CLI **login** command:

Azure CLICopy

```
az login
```

Azure CLI will typically launch your default browser to open the Azure sign in page.

If it doesn't work, follow the command-line instructions, and enter an authorization code in the [Enter Code](#) dialog box.

After a successful sign in, you'll be connected to your Azure subscription.

## 2. Create

You'll often need to create a new resource group before you create a new Azure service.

So we'll use resource groups as an example to show how to create Azure resources from the Azure CLI.

The Azure CLI **group create** command creates a resource group.

You need to specify a name and location.

The *name* parameter must be unique within your subscription.

The *location* parameter determines where the metadata for your resource group will be stored.

You use strings like "West US," "North Europe," or "West India" to specify the location.

Instead, you can use single-word equivalents, such as "westus," "northeurope," or "westindia."

The core syntax to create a resource group is:

Azure CLICopy

```
az group create --name <name> --location <location>
```

### 3. Verify installation

For many Azure resources, Azure CLI provides a **list** subcommand to get resource details.

For example, the Azure CLI **group list** command lists your Azure resource groups.

It's helpful to verify whether resource group creation was successful:

Azure CLICopy

```
az group list
```

To get more concise information, you can format the output as a simple table:

Azure CLICopy

```
az group list --output table
```

If you have several items in the group list, you can filter the return values by adding a **query** option using, for example, the following command:

Azure CLICopy

```
az group list --query "[?name == '<rg name>']"
```

#### Note

You format the query using **JMESPath**, which is a standard query language for JSON requests.

You can learn more about this filter language at <http://jmespath.org/>.

# Using Azure CLI in scripts

To use Azure CLI commands in scripts, you'll need to be aware of any issues around the shell or environment you use to run the script.

For example, in a bash shell, you can use the following syntax when setting variables:

Azure CLICopy

```
variable="value"  
variable=integer
```

If you use a PowerShell environment for running Azure CLI scripts, you'll need to use the following syntax for variables:

PowerShellCopy

```
$variable="value"  
$variable=integer
```

## Exercise - Run templates using Azure CLI

Completed 100 XP

- 4 minutes

### Important

To do these steps, you need an Azure subscription. If you don't have one already, you can create one by following the steps outlined on the [Create your Azure free](#).

## Steps

In the following steps, we'll deploy the template and verify the result using Azure CLI:

1. Create a resource group to deploy your resources to by running the following command:

Azure CLICopy

```
az group create --name <resource group name> --location <your nearest datacenter>
```

### Note

Check the available region for you [Choose the Right Azure Region for You](#). If you can't create in the nearest region, feel free to choose another one.

2. From Cloud Shell, run the **curl** command to download the template you used previously from GitHub:

BashCopy

```
curl https://raw.githubusercontent.com/Microsoft/PartsUnlimited/master/Labfiles/AZ-400T05_Implementing_Application_Infrastructure/M01/azuredeploy.json > C:\temp\azuredeploy.json
```

3. Validate the template by running the following command, replacing the values with your own:

Azure CLICopy

```
az deployment group validate \
--resource-group [sandbox resource group name] \
--template-file C:\temp\azuredeploy.json \
--parameters adminUsername=$USERNAME \
--parameters adminPassword=$PASSWORD \
--parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

4. Deploy the resource by running the following command, replacing the same values as earlier:

Azure CLICopy

```
az deployment group create \
--name MyDeployment \
--resource-group [sandbox resource group name] \
--template-file azuredeploy.json \
--parameters adminUsername=$USERNAME \
--parameters adminPassword=$PASSWORD \
--parameters dnsLabelPrefix=$DNS_LABEL_PREFIX
```

5. Obtain the IP address by running the following command:

Azure CLICopy

```
IPADDRESS=$(az vm show \
--name SimpleWinVM \
--resource-group [sandbox resource group name] \
--show-details \
--query [publicIps] \
--output tsv)
```

6. Run **curl** to access your web server and verify that the deployment and running of the custom script extension were successful:

BashCopy

```
curl $IPADDRESS
```

You'll have the following output:

HtmlCopy

```
<html><body><h2>Welcome to Azure! My name is SimpleWinVM.</h2></body></html>
```

#### Note

Don't forget to delete any resources you deployed to avoid incurring extra costs from them.

## Create Azure resources using Azure Resource Manager templates

### Why use Azure Resource Manager templates?

Completed 100 XP

- 3 minutes

Using Resource Manager templates will make your deployments faster and more repeatable.

For example, you no longer must create a VM in the portal, wait for it to finish, and then create the next VM. The Resource Manager takes care of the entire deployment for you.

Here are some other template benefits to consider:

- **Templates improve consistency.** Resource Manager templates provide a common language for you and others to describe your deployments. Despite the tool or SDK that you use to deploy the template, the template's structure, format, and expressions remain the same.
- **Templates help express complex deployments.** Templates enable you to deploy multiple resources in the correct order. For example, you wouldn't want to deploy a VM before creating an operating system (OS) disk or network interface. Resource Manager maps out each resource and its dependent resources and creates dependent resources first. Dependency

mapping helps ensure that the deployment is carried out in the correct order.

- **Templates reduce manual, error-prone tasks.** Manually creating and connecting resources can be time-consuming, and it's easy to make mistakes. The Resource Manager ensures that the deployment happens the same way every time.
- **Templates are code.** Templates express your requirements through code. Think of a template as a type of Infrastructure as Code that can be shared, tested, and versioned like any other piece of software. Also, because templates are code, you can create a record that you can follow. The template code documents the deployment. Also, most users maintain their templates under revision control, such as GIT. Its revision history also records how the template (and your deployment) has evolved when you change the template.
- **Templates promote reuse.** Your template can contain parameters that are filled in when the template runs. A parameter can define a username or password, a domain name, and other necessary items. Template parameters also enable you to create multiple versions of your infrastructure, such as staging and production, while still using the same template.
- Templates are linkable. You can link Resource Manager templates together to make the templates themselves modular. You can write small templates that define a solution and then combine them to create a complete system.

Azure provides many quickstart templates. You might use it as a base for your work.

## Explore template components

Completed 100 XP

- 4 minutes

Azure Resource Manager templates are written in **JSON**, which allows you to express data stored as an object (such as a virtual machine) in text.

A **JSON document** is essentially a collection of key-value pairs. Each key is a string that values can be:

- A string.
- A number.
- A Boolean expression.
- A list of values.

- An object (which is a collection of other key-value pairs).

A Resource Manager template can contain sections that are expressed using JSON notation but aren't related to the JSON language itself:

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

Let's review each of these sections in a little more detail.

## Parameters

This section is where you specify which values are configurable when the template runs.

For example, you might allow template users to set a username, password, or domain name.

Here's an example that illustrates two parameters: one for a virtual machine's (VMs) username and one for its password:

```
JSONCopy
"parameters": {
  "adminUsername": {
    "type": "string",
    "metadata": {
      "description": "Username for the Virtual Machine."
    }
  },
  "adminPassword": {
    "type": "securestring",
    "metadata": {
      "description": "Password for the Virtual Machine."
    }
}
```

## Variables

This section is where you define values that are used throughout the template.

Variables can help make your templates easier to maintain.

For example, you might define a storage account name one time as a variable and then use that variable throughout the template.

If the storage account name changes, you need only update the variable once.

Here's an example that illustrates a few variables that describe networking features for a VM:

JSONCopy

```
"variables": {  
    "nicName": "myVMNic",  
    "addressPrefix": "10.0.0.0/16",  
    "subnetName": "Subnet",  
    "subnetPrefix": "10.0.0.0/24",  
    "publicIPAddressName": "myPublicIP",  
    "virtualNetworkName": "MyVNET"  
}
```

## Functions

This section is where you define procedures that you don't want to repeat throughout the template.

Like variables, functions can help make your templates easier to maintain.

Here's an example that creates a function for creating a unique name to use when creating resources that have globally unique naming requirements:

JSONCopy

```
"functions": [  
    {  
        "namespace": "contoso",  
        "members": {  
            "uniqueName": {  
                "parameters": [  
                    {  
                        "name": "namePrefix",  
                        "type": "string"  
                    }  
                ]  
            }  
        }  
    }  
]
```

```
        }
    ],
    "output": {
        "type": "string",
        "value": "[concat(toLower(parameters('namePrefix')),  
uniqueString(resourceGroup().id))]"
    }
}
],
}
```

## Resources

This section is where you define the Azure resources that make up your deployment.

Here's an example that creates a public IP address resource:

JSONCopy

```
{
    "type": "Microsoft.Network/publicIPAddresses",
    "name": "[variables('publicIPAddressName')]",
    "location": "[parameters('location')]",
    "apiVersion": "2018-08-01",
    "properties": {
        "publicIPAllocationMethod": "Dynamic",
        "dnsSettings": {
            "domainNameLabel": "[parameters('dnsLabelPrefix')]"
        }
    }
}
```

Here, the type of resource is Microsoft.Network/publicIPAddresses.

The **name** is read from the variables section, and the **location**, or *Azure region*, is read from the **parameters** section.

Because resource types can change over time, apiVersion refers to the version of the resource type you want to use.

As resource types evolve, you can modify your templates to work with the latest features.

## Outputs

This section is where you define any information you'd like to receive when the template runs.

For example, you might want to receive your VM's IP address or fully qualified domain name (FQDN), the information you won't know until the deployment runs.

Here's an example that illustrates an output named **hostname**.

The FQDN value is read from the VM's public IP address settings:

```
JSONCopy
"outputs": {
    "hostname": {
        "type": "string",
        "value": "[reference(variables('publicIPAddressName')).dnsSettings.fqdn]"
    }
}
```

## Manage dependencies

Completed 100 XP

- 3 minutes

For any given resource, other resources might need to exist before you can deploy the resource.

For example, a Microsoft SQL Server must exist before attempting to deploy a SQL Database.

You can define this relationship by marking one resource as dependent on the other.

You define a dependency with the **dependsOn** element or by using the **reference** function.

Resource Manager evaluates the dependencies between resources and deploys them in their dependent order.

When resources aren't dependent on each other, the Resource Manager deploys them in parallel.

You only need to define dependencies for resources that are deployed in the same template.

## The dependsOn element

Within your template, the **dependsOn** element enables you to define one resource dependent on one or more other resources.

Its value can be a comma-separated list of resource names.

```
129      "type": "Microsoft.Compute/virtualMachines",
130      "name": "[variables('vmName')]",
131      "location": "[parameters('location')]",
132      "apiVersion": "2018-10-01",
133      "dependsOn": [
134          "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
135          "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
136      ],
```

## Circular dependencies

A **circular dependency** is a problem with dependency sequencing, resulting in the deployment going around in a loop and unable to continue.

As a result, the Resource Manager can't deploy the resources.

Resource Manager identifies circular dependencies during template validation.

If you receive an error stating that a circular dependency exists, evaluate your template to find whether any dependencies are unnecessary and can be removed.

If removing dependencies doesn't resolve the issue, you can move some deployment operations into child resources that are deployed after the resources with the circular dependency.

## Modularize templates

Completed 100 XP

- 4 minutes

When using Azure Resource Manager templates, it's best to modularize them by breaking them into individual components.

The primary methodology to use is by using linked templates.

It allows you to break the solution into targeted components and reuse those various elements across different deployments.

## Linked template

Add a deployment resource to your main template to link one template to another.

```
JSONCopy
"resources": [
  {
    "apiVersion": "2017-05-10",
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
      "mode": "Incremental",
      <link-to-external-template>
    }
  }
]
```

## Nested template

You can also nest a template within the main template, use the template property, and specify the template syntax.

It does somewhat aid modularization, but dividing up the various components can result in a sizeable main file, as all the elements are within that single file.

```
JSONCopy
"resources": [
  {
    "apiVersion": "2017-05-10",
    "name": "nestedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
      "mode": "Incremental",
      "template": {
        "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
        "contentVersion": "1.0.0.0",
        "resources": [
          {
            "type": "Microsoft.Resources/deployments"
          }
        ]
      }
    }
]
```

```

    "type": "Microsoft.Storage/storageAccounts",
    "name": "[variables('storageName')]",
    "apiVersion": "2015-06-15",
    "location": "West US",
    "properties": {
      "accountType": "Standard_LRS"
    }
  }
]
}
}
]

```

### Note

You can't use parameters or variables defined within the nested template itself for nested templates. You can only use parameters and variables from the main template.

The properties you provide for the deployment resource will vary based on linking to an external template or nesting an inline template within the main template.

## Deployments modes

When deploying your resources using templates, you have three options:

- **validate**. This option compiles the templates, validates the deployment, ensures the template is functional (for example, no circular dependencies), and correct syntax.
- **incremental mode (default)**. This option only deploys whatever is defined in the template. It doesn't remove or modify any resources that aren't defined in the template. For example, if you've deployed a VM via template and then renamed the VM in the template, the first VM deployed will remain after the template is rerun. It's the default mode.
- **complete mode**: Resource Manager deletes resources that exist in the resource group but isn't specified in the template. For example, only resources defined in the template will be present in the resource group after the template deploys. As a best practice, use this mode for production environments to achieve idempotency in your deployment templates.

When deploying with PowerShell, to set the deployment mode use the *Mode* parameter, as per the nested template example earlier in this topic.

### Note

As a best practice, use one resource group per deployment.

### Note

You can only use `incremental` deployment mode for both linked and nested templates.

## External template and external parameters

To link to an external template and parameter file, use `templateLink` and `parametersLink`.

When linking to a template, ensure that the Resource Manager service can access it.

For example, you can't specify a local file or a file only available on your local network.

You can only provide a Uniform Resource Identifier (URI) value that includes HTTP or HTTPS.

One option is to place your linked template in a storage account and use the URI for that item.

You can also provide the parameter inline. However, you can't use both inline parameters and a link to a parameter file.

The following example uses the `templateLink` parameter:

### JSONCopy

```
"resources": [
  {
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "properties": {
      "mode": "Incremental",
      "templateLink": {

"uri":"https://linkedtemplateek1store.blob.core.windows.net/linkedtemplates/linkedStorageAccount.json?sv=2018-03-28&sr=b&sig=d09p7XnbhGq56B0%2BSW3o9tX7E2WUDIk%2BpF1MTK2eFfs%3D&se=2018-12-31T14%3A32%3A29Z&sp=r"
    },
    "parameters": {
      "storageAccountName": {"value": "[variables('storageAccountName')]"},
      "location": {"value": "[parameters('location')]"}
    }
  }
]
```

```
},  
},
```

## Securing an external template

Although the linked template must be available externally, it doesn't need to be made available to the public.

Instead, you can add your template to a private storage account accessible to only the storage account owner, creating shared access signature (SAS) tokens to enable access during deployment.

You add that SAS token to the URI for the linked template.

Even though the token is passed in as a secure string, the linked template's URI, including the SAS token, is logged in the deployment operations.

To limit exposure, you can also set an expiration date for the token.

## Manage secrets in templates

Completed 100 XP

- 4 minutes

When passing a secure value (such as a password) as a parameter during deployment, you can retrieve the value from an Azure Key Vault.

Reference the Key Vault and secret in your parameter file.

The value is never exposed because you only reference its Key Vault ID.

The Key Vault can exist in a different subscription than the resource group you're deploying it to.

## Deploy a Key Vault and secret

To create a Key Vault and secret, use either Azure CLI or PowerShell.

To access the secrets inside this Key Vault from a Resource Manager deployment, the Key Vault property **enabledForTemplateDeployment** must be **true**.

## Using Azure CLI

The following code snippet is an example of how you can deploy a Key Vault and secret using Azure CLI:

PowerShellCopy

```
keyVaultName='{your-unique-vault-name}'  
resourceGroupName='{your-resource-group-name}'  
location='centralus'  
userPrincipalName='{your-email-address-associated-with-your-subscription}'  
  
# Create a resource group  
az group create --name $resourceGroupName --location $location  
  
# Create a Key Vault  
az keyvault create \  
  --name $keyVaultName \  
  --resource-group $resourceGroupName \  
  --location $location \  
  --enabled-for-template-deployment true  
az keyvault set-policy --upn $userPrincipalName --name $keyVaultName --secret-permissions set delete get list  
  
# Create a secret with the name, vmAdminPassword  
password=$(openssl rand -base64 32)  
echo $password  
az keyvault secret set --vault-name $keyVaultName --name 'vmAdminPassword' --value  
$password
```

## Enable access to the secret

Other than setting the Key Vault property **enabledForTemplateDeployment** to **true**, the user deploying the template must have Microsoft.KeyVault/vaults/deploy/action permission for the Key Vault scope.

Also including the resource group and Key Vault. The Owner and Contributor roles both grant this access.

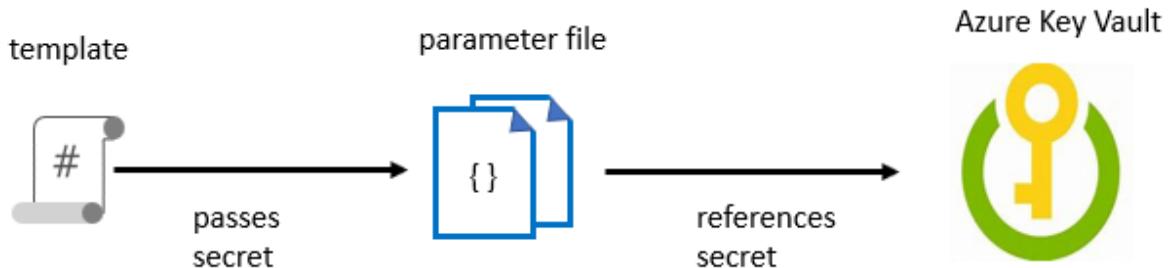
If you create the Key Vault, you're the owner, so you inherently have permission.

However, if the Key Vault is under a different subscription, the owner of the Key Vault must grant access.

## Reference a secret with static ID

The Key Vault is referenced in the parameter file and not the template.

The following image shows how the parameter file references the secret and passes that value to the template.



The following template deploys an SQL database that includes an administrator password.

The password parameter is set to a secure string. However, the template doesn't specify where that value comes from:

JSONCopy

```
{  
  "$schema": "https://schema.management.azure.com/schemas/2015-01-  
01/deploymentTemplate.json#",  
  "contentVersion": "1.0.0.0",  
  "parameters": {  
    "adminLogin": {  
      "type": "string"  
    },  
    "adminPassword": {  
      "type": "securestring"  
    },  
    "sqlServerName": {  
      "type": "string"  
    }  
  },  
  "resources": [  
    {  
      "name": "[parameters('sqlServerName')]",  
      "type": "Microsoft.Sql/servers",  
      "dependsOn": []  
    }  
  ]  
}
```

```

    "apiVersion": "2015-05-01-preview",
    "location": "[resourceGroup().location]",
    "tags": {},
    "properties": {
        "administratorLogin": "[parameters('adminLogin')]",
        "administratorLoginPassword": "[parameters('adminPassword')]",
        "version": "12.0"
    }
}
],
"outputs": {
}
}

```

Now you can create a parameter file for the preceding template. In the parameter file, specify a parameter that matches the parameter's name in the template.

For the parameter value, reference the secret from the Key Vault. You reference the secret by passing the resource identifier of the Key Vault and the secret's name.

The Key Vault secret must already exist in the following parameter file, and you provide a static value for its resource ID.

Copy this file locally, and set the subscription ID, vault name, and SQL server name:

JSONCopy

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "adminLogin": {
            "value": "exampleadmin"
        },
        "adminPassword": {
            "reference": {
                "keyVault": {
                    "id": "/subscriptions/<subscription-id>/resourceGroups/examplegroup/providers/Microsoft.KeyVault/vaults/<vault-name>"
                },
                "secretName": "examplesecret"
            }
        },
        "sqlServerName": {
            "value": "<your-server-name>"
        }
    }
}
```

You would need to deploy the template and pass the parameter file to the template.

For more information, use [Azure Key Vault to pass secure parameter values during deployment](#) for more details.

There are also details on this web page for reference to a secret with a dynamic ID.

# Deployments using Azure Bicep templates

Completed 100 XP

- 60 minutes

**Estimated time:** 60 minutes.

**Lab files:** none.

## Scenario

In this lab, you'll create an Azure Bicep template and modularize it using the Azure Bicep Modules concept. You'll then modify the main deployment template to use the module and finally deploy the all the resources to Azure.

## Objectives

After completing this lab, you'll be able to:

- Understand and create a Azure Bicep Templates.
- Create a reusable Bicep module for storage resources.
- Upload Linked Template to Azure Blob Storage and generate SAS token.
- Modify the main template to use the module.
- Modify the main template to update dependencies.
- Deploy all the resources to Azure using Azure Bicep Templates.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft account or a Microsoft Entra account with the Owner role in the Azure subscription and the Global Administrator role in the Microsoft Entra tenant associated with the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](#).
- [Visual Studio Code](#). This will be installed as part of the prerequisites for this lab.

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Author and deploy Azure Bicep templates.
- Exercise 2: Remove the Azure lab resources.

# Introduction to GitHub Packages

## Publish packages

Completed 100 XP

- 4 minutes

GitHub Packages use native package tooling commands to publish and install package versions.

Support for package registries

### Language

#### Package format

#### Package client

JavaScript

package.json

npm

Ruby

Gemfile

gem

Java

pom.xml

mvn

Java

build.gradle or build.gradle.kts

gradle

.NET

nupkg

dotnet CLI

N/A

Dockerfile

Docker

When creating a package, you can provide a description, installation and usage instructions, and other details on the package page. It helps people consuming the package understand how to use it and its purposes.

If a new package version fixes a security vulnerability, you can publish a security advisory to your repository.

**Tip**

You can connect a repository to more than one package. Ensure the README and description provide information about each package.

## Publishing a package

Using any supported package client, to publish your package, you need to:

1. Create or use an existing access token with the appropriate scopes for the task you want to accomplish: [Creating a personal access token](#). When you create a personal access token (PAT), you can assign the token to different scopes depending on your needs. See "[About permissions for GitHub Packages](#)".
2. Authenticate to GitHub Packages using your access token and the instructions for your package client.
3. Publish the package using the instructions for your package client.

Choose your package, and check how to authenticate and publish: [Working with a GitHub Packages registry](#). You'll see below examples for NuGet and npm.

NuGet registry

You can authenticate to GitHub Packages with the dotnet command-line interface (CLI).

Create a *nuget.config* file in your project directory and specify GitHub Packages as a source under packageSources.

#### XMLCopy

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="github" value="https://nuget.pkg.github.com/OWNER/index.json" />
  </packageSources>
  <packageSourceCredentials>
    <github>
      <add key="Username" value="USERNAME" />
      <add key="ClearTextPassword" value="TOKEN" />
    </github>
  </packageSourceCredentials>
</configuration>
```

#### Note

Replace *USERNAME* with the name of your personal account on GitHub, **TOKEN** with your PAT, and **OWNER** with the name of the user or organization account that owns your project's repository.

You can publish a package authenticating with a *nuget.config* file, or using the --api-key command-line option with your GitHub PAT.

#### dotnetCopy

```
dotnet nuget push "bin/Release/OctocatApp.1.0.0.nupkg" --api-key YOUR_GITHUB_PAT --
source "github"
```

#### npm registry

You can authenticate using npm by either editing your per-user *~/.npmrc* file to include your PAT or by logging in to npm on the command line using your username and personal access token.

Edit your *~/.npmrc* file for your project to include the following line:

```
//npm.pkg.github.com/:_authToken=TOKEN
```

Create a new *~/.npmrc* file if one doesn't exist.

If you prefer to authenticate by logging in to npm, use the `npm login` command.

```
$ npm login --scope=@OWNER --registry=https://npm.pkg.github.com
```

Username: USERNAME Password: TOKEN Email: PUBLIC-EMAIL-ADDRESS

### Note

Replace *USERNAME* with your GitHub username, *TOKEN* with your PAT, and *PUBLIC-EMAIL-ADDRESS* with your email address.

To publish your npm package, see [Working with the npm registry - GitHub Docs](#).

After you publish a package, you can view the package on GitHub. See "[Viewing packages](#)".

For an example package page, see [Codertocat/hello-world-npm \(github.com\)](#).

For more information, see:

- [Working with the Container registry](#).
- [Working with a GitHub Packages registry](#).
- [About GitHub Security Advisories](#).
- [Working with the NuGet registry](#).
- [Working with the npm registry](#).

## Install a package

Completed 100 XP

- 3 minutes

You can install any package you have permission to view from GitHub Packages and use the package as a dependency in your project.

You can search for packages globally across all of GitHub or within a particular organization. For details, see [Searching for packages](#).

After you find a package, read the package's installation and description instructions on the package page.

You can install a package using any supported package client following the same general guidelines.

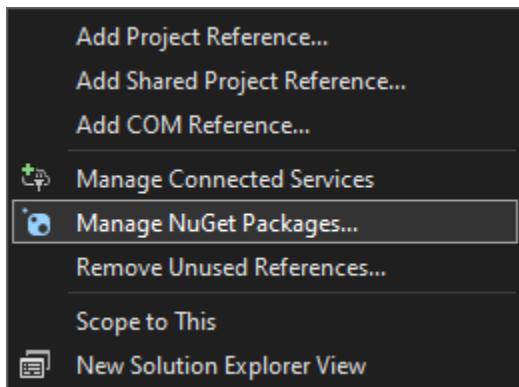
1. Authenticate to GitHub Packages using the instructions for your package client.

2. Install the package using the instructions for your package client.

## NuGet

To use NuGet packages from GitHub Packages, you must add dependencies to your `.csproj` file. For more information on using a `.csproj` file in your project, see ["Working with NuGet packages"](#).

If you're using Visual Studio, expand your Solution -> Project -> Right-click on Dependencies -> Manage NuGet Packages...



You can browse, install and update dependencies from multiple registries. For more information, see [Create and remove project dependencies](#).

A screenshot of the NuGet Package Manager interface. The top navigation bar shows 'Browse' (selected), 'Installed' (with 4 updates), and 'Updates 4'. Below the navigation is a search bar with 'Search (Ctrl+L)', a filter dropdown with 'P' and 'U', and a checkbox for 'Include prerelease'. The main area displays three installed packages:

- Microsoft.Identity.Web.UI** by Microsoft (version 1.24.0). Description: This package enables UI for ASP.NET Core Web apps that use Microsoft.Identity.Web.
- Microsoft.EntityFrameworkCore.SqlServer** by Microsoft (version 6.0.4). Description: Microsoft SQL Server database provider for Entity Framework Core.
- Microsoft.EntityFrameworkCore** by Microsoft (version 6.0.4). Description: Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.

Or, you can update your `.csproj` file directly.

1. Authenticate to GitHub Packages.
2. Add **ItemGroup** and configure the **PackageReference** field in the `.csproj` project file.

Copy

```
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>netcoreapp3.0</TargetFramework>
        <PackageId>OctocatApp</PackageId>
        <Version>1.0.0</Version>
        <Authors>Octocat</Authors>
        <Company>GitHub</Company>
        <PackageDescription>This package adds an
Octocat!</PackageDescription>

    <RepositoryUrl>https://github.com/OWNER/REPOSITORY</RepositoryUrl>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="OctokittenApp" Version="12.0.2" />
    </ItemGroup>
</Project>
```

**Note**

Replace the **OctokittenApp** package with your package dependency and **1.0.0** with the version you want to use.

3. Install the packages with the restore command.

npm

You need to add the `.npmrc` file to your project to install packages from GitHub Packages.

1. Authenticate to GitHub Packages.
2. In the same directory as your `package.json` file, create or edit a `.npmrc` file.
3. Include a line specifying GitHub Packages URL and the account owner.

```
@OWNER:registry=https://npm.pkg.github.com
```

**Note**

Replace **OWNER** with the name of the user or organization account.

4. Add the `.npmrc` file to the repository. See "[Adding a file to a repository](#)".
5. Configure `package.json` in your project to use the package you're installing.

JSONCopy

```
{  
  "name": "@my-org/server",  
  "version": "1.0.0",  
  "description": "Server app that uses the @octo-org/octo-app package",  
  "main": "index.js",  
  "author": "",  
  "license": "MIT",  
  "dependencies": {  
    "@octo-org/octo-app": "1.0.0"  
  }  
}
```

6. Install the package.

7. (Optional) If you need to install packages from other organizations, you can add extra lines to your `.npmrc` file.

For more information, see:

- [Working with a GitHub Packages registry](#).
- [Working with the NuGet registry](#).
- [Working with the npm registry](#).

## Delete and restore a package

Completed 100 XP

- 3 minutes

You can delete it on GitHub if you have the required access:

- An entire private package.
- If there aren't more than 5000 downloads of any version of the package, an entire public package.
- A specific version of a private package.
- A specific version of a public package if the package version doesn't have more than 5000 downloads.

For packages that inherit their access permissions from repositories, you can delete a package if you have admin permissions to the repository.

You can also restore an entire package or package version, if:

- You restore the package within 30 days of its deletion.
- The same package namespace is still available and not used for a new package.

You can use the REST API to manage your packages. For more information, see the "[GitHub Packages API](#)".

## Deleting a package version

You can delete a package version:

- Deleting a version of a repository-scoped package on GitHub.
- Deleting a version of a repository-scoped package with GraphQL.
- Deleting a version of a user-scoped package on GitHub.
- Deleting a version of an organization-scoped package on GitHub.

To delete a version of a repository-scoped package, you must have admin permissions to the repository that owns the package.

1. On GitHub.com, navigate to the main page of the repository.
2. To the right of the list of files, click Packages.
3. Search for and select your package.
4. In the top right of your package's landing page, click Package settings.
5. On the left, click Manage versions.
6. To the right of the version you want to delete, click and select Delete version.
7. To confirm the deletion, type the package name and click I understand the consequences, delete this version.

## Deleting an entire package

You can delete a package version:

- Deleting an entire repository-scoped package on GitHub.
- Deleting an entire user-scoped package on GitHub.
- Deleting an entire organization-scoped package on GitHub.

To delete an entire repository-scoped package, you must have admin permissions to the repository that owns the package.

1. On GitHub.com, navigate to the main page of the repository.
2. To the right of the list of files, click Packages.
3. Search for and select your package.
4. In the top right of your package's landing page, click Package settings.
5. Under "Danger Zone," click Delete this package.

6. Review the confirmation message, enter your package name, click I understand, and delete this package.

## Restoring a package version

You can restore a package version from your package's landing page.

1. Navigate to your package's landing page.
2. On the right, click Package settings.
3. On the left, click Manage versions.
4. Use the "Versions" drop-down menu and select Deleted on the top right.
5. Next to the deleted package version you want to restore, click Restore.
6. To confirm, click I understand the consequences, restore this version.

For more information, see:

- [Deleting and restoring a package - GitHub Docs](#).
- [Working with a GitHub Packages registry](#).
- [Working with the NuGet registry](#).
- [Working with the npm registry](#).
- [Required permissions](#).

## Explore package access control and visibility

Completed 100 XP

- 2 minutes

You can only use granular permissions with the Container registry (scoped to a personal user or organization account). GitHub.com doesn't support granular permissions in other package registries, like the npm registry.

You can change the package access control and visibility separately from the repository.

For more information about permissions, see "[About permissions for GitHub Packages](#)".

## Container images visibility and access permissions

If you have admin permissions to a container image, you can set the access permissions for the container image to private or public.

As an admin, you can also grant access permissions for a container image separate from your set permissions at the organization and repository levels.

You can give any person an access role for container images published and owned by a personal account.

For container images published and owned by an organization, you can provide any person or team in the organization an access role.

## Permission

### Access description

**read**

Can download the package. Can read package metadata.

**write**

Can upload and download this package. Can read and write package metadata.

**admin**

Can upload, download, delete, and manage this package. Can read and write package metadata. Can grant package permissions.

For more information about the package's access control and visibility configuration, see [Configuring a package's access control and visibility - GitHub Docs](#).

# Implement a versioning strategy

## Understand versioning of artifacts

Completed 100 XP

- 1 minute

It's proper software development practice to indicate changes to code with the introduction of an increased version number.

However small or large a change, it requires a new version. A component and its package can have independent versions and versioning schemes.

The versioning scheme can differ per package type. Typically, it uses a scheme that can indicate the kind of change that is made.

Most commonly, it involves three types of changes:

- **Major change** Major indicates that the package and its contents have changed significantly. It often occurs at the introduction of a new version of the package. It can be a redesign of the component. Major changes aren't guaranteed to be compatible and usually have breaking changes from older versions. Major changes might require a large amount of work to adopt the consuming codebase to the new version.
- **Minor change** Minor indicates that the package and its contents have extensive modifications but are smaller than a major change. These changes can be backward compatible with the previous version, although they aren't guaranteed to be.
- **Patch** A patch or revision is used to indicate that a flaw, bug, or malfunctioning part of the component has been fixed. Usually, it's a backward-compatible version compared to the previous version.

How artifacts are versioned technically varies per package type. Each type has its way of indicating the version in the metadata.

The corresponding package manager can inspect the version information. The tooling can query the package feed for packages and the available versions.

Additionally, a package type might have its conventions for versioning and a particular versioning scheme.

See also [Publish to NuGet feeds](#).

# Explore semantic versioning

Completed 100 XP

- 1 minute

One of the predominant ways of versioning is the use of semantic versioning.

It isn't a standard but does offer a consistent way of expressing the intent and semantics of a particular version.

It describes a version for its backward compatibility with previous versions.

Semantic versioning uses a three-part version number and an extra label.

The version has the form of `Major.Minor.Patch` corresponds to the three types of changes covered in the previous section.

Examples of versions using the semantic versioning scheme are `1.0.0` and `3.7.129`. These versions don't have any labels.

For prerelease versions, it's customary to use a label after the regular version number.

A label is a textual suffix separated by a hyphen from the rest of the version number.

The label itself can be any text describing the nature of the prerelease.

Examples of these are `rc1`, `beta27`, and `alpha`, forming version numbers like `1.0.0-rc1` is a prerelease for the upcoming `1.0.0` version.

Prereleases are a common way to prepare for the release of the label-less version of the package.

Early adopters can take a dependency on a prerelease version to build using the new package.

Generally, using a prerelease version of packages and their components for released software isn't a good idea.

It's good to expect the impact of the new components by creating a separate branch in the codebase and using the prerelease version of the package.

Chances are that there will be incompatible changes from a prerelease to the final version.

See also [Semantic Versioning 2.0.0](#).

## Examine release views

Completed 100 XP

- 2 minutes

When building packages from a pipeline, the package needs to have a version before being consumed and tested.

Only after testing is the quality of the package known.

Since package versions can't and shouldn't be changed, it becomes challenging to choose a specific version beforehand.

Azure Artifacts recognizes the quality level of packages in its feeds and the difference between prerelease and release versions.

It offers different views on the list of packages and their versions, separating these based on their quality level.

It fits well with the use of semantic versioning of the packages for predictability of the intent of a particular version.

Still, its extra metadata from the Azure Artifacts feed is called a descriptor.

Feeds in Azure Artifacts have three different views by default. These views are added when a new feed is created. The three views are:

- **Local.** The @Local view contains all release and prerelease packages and the packages downloaded from upstream sources.
- **Prerelease.** The @Prerelease view contains all packages that have a label in their version number.

- **Release.** The @Release view contains all packages that are considered official releases.

## Using views

You can use views to offer help consumers of a package feed filter between released and unreleased versions of packages.

Essentially, it allows a consumer to make a conscious decision to choose from released packages or opt-in to prereleases of a certain quality level.

By default, the @Local view is used to offer the list of available packages. The format for this URI is:

URLCopy

```
https://pkgs.dev.azure.com/{yourteamproject}/_packaging/{feedname}/nuget/v3/index.json
```

When consuming a package feed by its URI endpoint, the address can have the requested view included. For a specific view, the URI includes the name of the view, which changes to be:

URLCopy

```
https://pkgs.dev.azure.com/{yourteamproject}/_packaging/{feedname}@{Viewname}/nuget/v3/index.json
```

The tooling will show and use the packages from the specified view automatically.

Tooling may offer an option to select prerelease versions, such as shown in this Visual Studio 2017 NuGet dialog. It doesn't relate or refer to the @Prerelease view of a feed. Instead, it relies on the presence of prerelease labels of semantic versioning to include or exclude packages in the search results.

See also:

- [Views on Azure DevOps Services feeds.](#)
- [Communicate package quality with prerelease and release views.](#)

# Promote packages

Completed 100 XP

- 1 minute

Azure Artifacts has the notion of promoting packages to views to indicate that a version is of a certain quality level.

By selectively promoting packages, you can plan when packages have a certain quality and are ready to be released and supported by the consumers.

You can promote packages to one of the available views as the quality indicator.

Release and Prerelease's two views might be sufficient, but you can create more views when you want finer-grained quality levels if necessary, such as alpha and beta.

**Packages** will always show in the Local view, but only in a particular view after being promoted.

Depending on the URL used to connect to the feed, the available packages will be listed.

**Upstream** sources will only be evaluated when using the @Local view of the feed.

After they've been downloaded and cached in the @Local view, you can see and resolve the packages in other views after being promoted to it.

It's up to you to decide how and when to promote packages to a specific view.

This process can be automated by using an Azure Pipelines task as part of the build pipeline.

Packages that have been promoted to a view won't be deleted based on the retention policies.

## Exercise - Promote a package

Completed 100 XP

- 2 minutes

**Important**

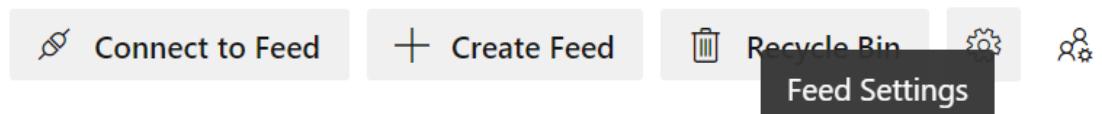
Have access to an existing Azure DevOps project and the connected package feed from the previous demo [Create a package feed](#).

## Steps to demonstrate the views on package feeds in Azure Artifacts

1. Go to dev.azure.com and open your team project.
2. Open **Artifacts** and select the feed **PartsUnlimited**.

The screenshot shows the Azure Artifacts interface. On the left, there is a sidebar with icons for Overview, Boards, Repos, Pipelines, Test Plans, and Artifacts. The 'Artifacts' icon is highlighted with a blue selection bar. To the right, a main panel displays the 'PartsUnlimited' feed settings. At the top, there is a dropdown menu set to 'PartsUnlimited'. Below it is a 'Feeds' section with a 'Filter feeds' search bar. Under 'Active Feeds', there is a 'Project scoped feeds' section containing 'PartsUnlimited'. Below this are sections for 'Organization scoped feeds' and 'DevLabsPackages'.

3. Go to **Artifacts** and click **Feed Settings**.



4. Open the **Views** tab. By default, there will be three views. Local: includes all packages in the feed and all cached from upstream sources. Prerelease and Release. In the **Default view** column is a check behind Local. It's the default view that will always be used.

## Steps to use the release view instead

1. Open Visual Studio and open NuGet Package Manager.
2. Click the settings wheel and check the source address for the PartsUnlimited feed.  
If you want to use a different view than the local view, you need to include that in the Source URL of your feed by adding @Release.
3. Add @Release to the source URL .../PartsUnlimited@Release/nuget/... And click **Update**.
4. **Refresh** the Browse tab. You'll see there are **No packages found** in the Release feed. Before any packages appear, you need to promote them.

## Steps to promote packages to views

1. Go back to your feed in Azure Artifacts.
2. Click on the created NuGet Package **PartsUnlimited.Security**.
3. Click **Promote**.
4. Select the feed you want to use, in this case, **PartsUnlimited@Release**, and Promote.
5. Go back to the **Overview**. If we look again at our package in the feed, you'll notice a Release tag is now associated with this package.
6. Go back to Visual Studio, **NuGet Package Manager**.
7. **Refresh** the Browse tab. You'll see that your version is promoted to this view.
8. Select **PartsUnlimited.Security** and click **Update** and **OK**. The latest version of this package is now used.

## Explore best practices for versioning

100 XP

- 1 minute

There are several best practices to effectively use versions, views, and promotion flow for your versioning strategy.

Here's a couple of suggestions:

- Have a documented versioning strategy.
- Adopt SemVer 2.0 for your versioning scheme.
- Each repository should only reference one feed.
- On package creation, automatically publish packages back to the feed.

It's good to adopt a best practice yourself and share these with your development teams.

It can be made part of the Definition of Done for work items related to publishing and consuming packages from feeds.

See also [Best practices for using Azure Artifacts](#).

## Exercise - Push from the pipeline

Completed 100 XP

- 2 minutes

### Important

In your Azure DevOps PartsUnlimited project, prepare two builds pipelines (used in previous demos)

- Build pipeline **PartsUnlimited Security Package**.
  - .NET Core build task (Restore, build, push)
  - Enable CI trigger
- Build pipeline **PartsUnlimited E2E**.
  - ASP.NET web application type
  - NuGet restore task

## Steps to build and push the NuGet package

1. Edit the build pipeline **PartsUnlimited Security Package**.
  - dotnet restore
  - dotnet build
  - dotnet push
    - Use the command `nuget push` and specify path `**/*.nupkg`.
    - Select target feed PartsUnlimited.
2. Start a new build and select agent pool.

The build has succeeded, but you'll see the final step **dotnet push** fails. The reason for this failure can be found in the log information.

3. Open the log information.

It shows the feed already contains the **PartsUnlimited.Security 1.0.0**. We go back to the Visual Studio project to see what is happening.

4. Open the source code for the PartsUnlimited package in Visual Studio in a different solution.
  - Open the **Project Properties**.
  - Go to the package tab.

Look at the Package version. We see that the version is still 1.0.0. Packages are immutable.

As soon as a package is published to a feed, there can never be another package with the same version.

We need to upgrade the version to a new one that uses the major, minor, and the changed patch version.

5. Change the patch version: 1.0.1. Make a small edit to the code for illustrative purposes, and check in the new code.
6. Change the **exception type** check-in, commit, and push the new code. We go back to the Azure DevOps pipeline. Since there's a trigger on the code we just changed, the build will automatically start.
7. Go back to build pipeline for **PartsUnlimited Security Package**.

As you see, the build is triggered and completed successfully, including the push to the NuGet feed.

Since there was no version conflict, we successfully pushed the new version to the feed.

8. Open **Artifacts** and show the feed.

Since there was a successful build for the entire web application, you can see that the PartsUnlimited feed now also includes all the downloaded upstream packages from the NuGet.org source.

9. Scroll down and click on the **PartsUnlimited.Security 1.0.1** package. By clicking on it, we can inspect the details and the versions.
10. Edit the build pipeline **PartsUnlimited E2E**.
11. Click **NuGet restore**.

There is a second pipeline that builds the complete web application. It is an ASP.NET web application build.

The NuGet restore task is also configured to consume packages from the PartsUnlimited feed.

Because PartsUnlimited has an upstream source for NuGet.org, we don't have to **Use packaged from NuGet.org** explicitly. You can uncheck this box.

# Migrate consolidating and secure artifacts

## Identify existing artifact repositories

Completed 100 XP

- 1 minute

An **artifact** is a deployable component of your application. Azure Pipelines can work with a wide variety of artifact sources and repositories.

When you're creating a release pipeline, you need to link the required artifact sources. Often, it will represent the output of a build pipeline from a continuous integration system like Azure Pipelines, Jenkins, or TeamCity.

The artifacts that you produce might be stored in source control, like Git or Team Foundation version control. But you might also be using package management tools when you get repositories.

When you need to create a release, you must specify which version of the artifacts is required. By default, the release pipeline will choose the latest version of the artifacts. But you might not want that.

For example, you might need to choose a specific branch, a specific build version, or perhaps you need to specify tags.

Azure Artifacts are one of the services that's part of Azure DevOps. Using it can eliminate the need to manage file shares or host private package services.

It lets you share code easily by allowing you to store Maven, npm, or NuGet packages together, cloud-hosted, indexed and matched.

Now, while we can do so, there's also no need to store your binaries in Git. You can keep them directly using universal packages. It's also a great way to protect your packages. Azure Artifacts provide universal artifact management from Maven, npm, and NuGet.

And sharing packages, you can easily access all of your artifacts in builds and releases because it integrates naturally with Azure Pipelines and its CI/CD tooling, along with versioning and testing.

# Migrate and integrating artifact repositories

Completed 100 XP

- 1 minute

While you can continue to work with your existing artifact repositories in their current locations when using Azure Artifacts, there are advantages to migrating them.

## NuGet and other packages

Azure Artifacts provides hosted NuGet feeds as a service.

Using this service, you can often eliminate the dependencies on on-premises resources such as file shares and locally hosted instances of NuGet.Server.

The feeds can also be consumed by any Continuous Integration system that supports authenticated NuGet feeds.

## Walkthroughs

For details on how to integrate NuGet, npm, Maven, Python, and Universal Feeds, see the following walkthroughs:

- Get started with NuGet packages in Azure DevOps Services and TFS.
- Use npm to store JavaScript packages in Azure DevOps Services or TFS.
- Get started with Maven packages in Azure DevOps Services and TFS.
- Get started with Python packages in Azure Artifacts.
- Publish and then download a Universal Package.

## Secure access to package feeds

Completed 100 XP

- 2 minutes

## Trusted sources

Package feeds are a trusted source of packages. The offered packages will be consumed by other codebases and used to build software that needs to be secure.

Imagine what would happen if a package feed would offer malicious components in its packages.

Each consumer would be affected when installing the packages onto its development machine or build server.

It also happens at any other device that will run the end product, as the malicious components will be executed as part of the code.

Usually, the code runs with high privileges, giving a large security risk if any packages cannot be trusted and might contain unsafe code.

## Securing access

Package feeds must be secured for access by authorized accounts, so only verified and trusted packages are stored there.

None should push packages to a feed without the proper role and permissions.

It prevents others from pushing malicious packages. It still assumes that the persons who can push packages will only add safe and secure packages.

Especially in the open-source world, it's done by the community. A package source can further guard its feed with the use of security and vulnerability scan tooling.

Additionally, consumers of packages can use similar tooling to do the scans themselves.

## Securing availability

Another aspect of security for package feeds is about the public or private availability of the packages.

The feeds of public sources are available for anonymous consumption.

Private feeds have restricted access most of the time.

It applies to the consumption and publishing of packages. Private feeds will allow only users in specific roles or teams access to its packages.

Package feeds need to have secure access for different kinds of reasons.

The access should involve allowing:

- **Restricted access for consumption** Whenever a particular audience should only consume a package feed and its packages, it's required to restrict its access. Only those allowed access will consume the packages from the feed.
- **Restricted access for publishing** Secure access is required to restrict who can publish so feeds and unauthorized or untrusted persons and accounts can't modify their packages.

## Examine roles

Completed 100 XP

- 1 minute

Azure Artifacts has four different roles for package feeds. These are incremental in the permissions they give.

The roles are in incremental order:

- Reader: Can list and restore (or install) packages from the feed.
- Collaborator: Can save packages from upstream sources.
- Contributor: Can push and unlist packages in the feed.
- Owner: has all available permissions for a package feed.

When creating an Azure Artifacts feed, the Project Collection Build Service is given contributor rights by default.

This organization-wide build identity in Azure Pipelines can access the feeds it needs when running tasks.

If you changed the build identity to be at the project level, you need to give that identity permissions to access the feed.

Any contributors to the team project are also contributors to the feed.

Project Collection Administrators and administrators of the team project, plus the feed's creator, are automatically made owners of the feed.

The roles for these users and groups can be changed or removed.

## Examine permissions

Completed 100 XP

- 1 minute

The feeds in Azure Artifacts require permission to the various features it offers. The list of permissions consists of increasing privileged operations.

The list of privileges is as follows:

| Permission                          | Reader | Collaborator | Contributor | Owner |
|-------------------------------------|--------|--------------|-------------|-------|
| List and restore/install packages   | ✓      | ✓            | ✓           | ✓     |
| Save packages from upstream sources |        | ✓            | ✓           | ✓     |
| Push packages                       |        |              | ✓           | ✓     |
| Unlist/deprecate packages           |        |              | ✓           | ✓     |
| Delete/unpublish package            |        |              |             | ✓     |
| Edit feed permissions               |        |              |             | ✓     |
| Rename and delete feed              |        |              |             | ✓     |

You can assign users, teams, and groups to a specific role for each permission, giving the permissions corresponding to that role.

You need to have the Owner role to do so. Once an account has access to the feed from the permission to list and restore packages, it's considered a Feed user.

## DevOpsCertificationFeed > Feed settings

Feed details Permissions Views Upstream sources | + Add users/groups Delete ...

Filter by User/Group

| User/Group   | Role        |
|--|-------------|
| [DevOpsCertification-Course-MS]\Project Administrators | Owner       |
| Project Collection Build Service                       | Contributor |
| [DevOpsCertification-Course-MS]\Contributors           | Contributor |

Like permissions and roles for the feed itself, there are extra permissions for access to the individual views.

Any feed user has access to all the views, whether the default views of @Local, @Release, @Prerelease or newly created ones.

When creating a feed, you can choose whether the feed is visible to people in your Azure DevOps organization or only specific people.

Visibility - Who can use your feed

-  People in Contoso - Members of your organization can view the packages in your feed
-  Specific people - Only people you give access to will be able to view this feed

See also: [Secure and share packages using feed permissions](#).

## Examine authentication

Completed 100 XP

- 1 minute

Azure DevOps users will authenticate against Microsoft Entra ID when accessing the Azure DevOps portal.

After being successfully authenticated, they won't have to provide any credentials to Azure Artifacts itself. The roles for the user, based on its identity, or team and group membership, are for authorization.

When access is allowed, the user can navigate to the Azure Artifacts section of the team project.

The authentication from Azure Pipelines to Azure Artifacts feeds is taken care of transparently. It will be based upon the roles and their permissions for the build identity.

The previous section on Roles covered some details on the required roles for the build identity.

The authentication from inside Azure DevOps doesn't need any credentials for accessing feeds by itself.

However, when accessing secured feeds outside Azure Artifacts, such as other package sources, you most likely must provide credentials to authenticate to the feed manager.

Each package type has its way of handling the credentials and providing access upon authentication. The command-line tooling will provide support in the authentication process.

For the build tasks in Azure Pipelines, you'll provide the credentials via a Service connection.

# Understand package management

## Explore packages

Completed 100 XP

- 4 minutes

Packages are used to define the components you rely on and depend upon in your software solution.

They provide a way to store those components in a well-defined format with metadata to describe them.

### What is a package?

A package is a formalized way of creating a distributable unit of software artifacts that can be consumed from another software solution.

The package describes the content it contains and usually provides extra metadata, and the information uniquely identifies the individual packages and is self-descriptive.

It helps to better store packages in centralized locations and consume the contents of the package predictably.

Also, it enables tooling to manage the packages in the software solution.

### Types of packages

Packages can be used for different kinds of components.

The type of components you want to use in your codebase differ for the different parts and layers of the solution you're creating.

The range from frontend components, such as JavaScript code files, to backend components like .NET assemblies or Java components, complete self-contained solutions, or reusable files in general.

Over the past years, the packaging formats have changed and evolved. Now there are a couple of de facto standard formats for packages.

- **NuGet** packages (pronounced "new get") are a standard used for .NET code artifacts. It includes .NET assemblies and related files, tooling, and sometimes only metadata. NuGet defines the way packages are created, stored, and consumed. A NuGet package is essentially a compressed folder structure with files in ZIP format and has the .nupkg extension. See also [An introduction to NuGet](#).
- **npm** package is used for JavaScript development. It originates from node.js development, where it's the default packaging format. An npm package is a file or folder containing JavaScript files and a package.json file describing the package's metadata. For node.js, the package usually includes one or more modules that can be loaded once the package is consumed. See also [About packages and modules](#).
- **Maven** is used for Java-based projects. Each package has a Project Object Model file describing the project's metadata and is the basic unit for defining a package and working with it.
- **PyPi** The Python Package Index, abbreviated as PyPi and known as the Cheese Shop, is the official third-party software repository for Python.
- **Docker** packages are called images and contain complete and self-contained deployments of components. A Docker image commonly represents a software component that can be hosted and executed by itself without any dependencies on other images. Docker images are layered and might be dependent on other images as their basis. Such images are referred to as base images.

## Understand package feeds

Completed 100 XP

- 3 minutes

Packages should be stored in a centralized place for distribution and consumption to take dependencies on the components it contains.

The centralized storage for packages is commonly called a package feed. There are other names in use, such as repository or registry.

We'll refer to all of these as package feeds unless it's necessary to use the specific name for clarity. Each package type has its type of feed.

Put another way. One feed typically contains one type of packages. There are NuGet feeds, npm feeds, Maven repositories, PyPi feed, and Docker registries.

Package feeds offer versioned storage of packages. A particular package can exist in multiple versions in the feed, catering for consumption of a specific version.

## Private and public feeds

The package feeds are centralized and available for many different consumers.

Depending on the package, purpose, and origin, it might be generally available or to a select audience.

Typically, open-source projects for applications, libraries, and frameworks are shared with everyone and publically available.

The feeds can be exposed in public or private to distinguish in visibility. Anyone can consume public feeds.

There might be reasons why you don't want your packages to be available publicly.

It could be because it contains intellectual property or doesn't make sense to share with other software developers.

Components developed for internal use might be available only to the project, team, or company that developed it.

In such cases, you can still use packages for dependency management and choose to store the package in a private package feed.

Private feeds can only be consumed by those who are allowed access.

## Explore package feed managers

Completed 100 XP

- 2 minutes

Each of the package types has a corresponding manager that takes care of one or more of the following aspects of package management:

- Installation and removal of local packages
- Pushing packages to a remote package feed

- Consuming packages from a remote package feed
- Searching feeds for packages

The package manager has cross-platform command-line interface (CLI) tools to manage the local packages and feeds that host the packages. This CLI tooling is part of a local install on a development machine.

## Choosing tools

The command-line nature of the tooling offers the ability to include it in scripts to automate package management. Ideally, one should use the tooling in build and release pipelines for component creating, publishing, and consuming packages from feeds.

Additionally, developer tooling can have integrated support for working with package managers, providing a user interface for the raw tooling. Examples of such tooling are Visual Studio 2017, Visual Studio Code, and Eclipse.

# Explore common public package sources

Completed 100 XP

- 3 minutes

The various package types have a standard source that is commonly used for public use.

It's a go-to place for developers to find and consume publically available components as software dependencies. These sources are package feeds.

## Public

In general, you'll find that publically available package sources are free to use.

Sometimes they have a licensing or payment model for consuming individual packages or the feed itself.

These public sources can also be used to store packages you've created as part of your project.

It doesn't have to be open-source, although it is in most cases.

Public and free package sources that offer feeds at no expense will usually require that you make the packages you store publically available as well.

Expand table

| Package type | Package source       | URL   |
|--------------|----------------------|---|
| NuGet        | NuGet Gallery        | <a href="https://nuget.org">https://nuget.org</a>               |
| npm          | npmjs                | <a href="https://npmjs.org">https://npmjs.org</a>               |
| Maven        | Maven                | <a href="https://search.maven.org">https://search.maven.org</a> |
| Docker       | Docker Hub           | <a href="https://hub.docker.com">https://hub.docker.com</a>     |
| Python       | Python Package Index | <a href="https://pypi.org">https://pypi.org</a>                 |

The table above doesn't contain an extensive list of all public sources available.

There are other public package sources for each of the types.

## Explore self-hosted and SaaS based package sources

Completed 100 XP

- 3 minutes

The following private package sources will give you a starting point for finding the most relevant feeds.

## Private

Private feeds can be used where packages should be available to a select audience.

The main difference between public and private feeds is the need for authentication.

Public feeds can be anonymously accessible and optionally authenticated.

Private feeds can be accessed only when authenticated.

There are two options for private feeds:

- **Self-hosting** Some of the package managers are also able to host a feed. One can host the required solution to offer a private feed using on-premises or private cloud resources.
- **SaaS services** A variety of third-party vendors and cloud providers offer software-as-a-service feeds that can be kept private. It typically requires a consumption fee or a cloud subscription.

The following table contains a non-exhaustive list of self-hosting options and SaaS offerings to host private package feeds for each type covered.

Expand table

| Package type | Self-hosted private feed     | SaaS private feed   |
|--------------|------------------------------|---|
| NuGet        | NuGet server                 | Azure Artifacts, MyGet  |
| npm          | Sinopia, cnpmjs, Verdaccio   | npmjs, MyGet, Azure Artifacts   |
| Maven        | Nexus, Artifactory, Archivia | Azure Artifacts, Bintray, JitPack                                       |
| Docker       | Portus, Quay, Harbor         | Docker Hub, Azure Container Registry, Amazon Elastic Container Registry |
| Python       | PyPI Server                  | Gemfury   |

## Consume packages

Completed 100 XP

- 4 minutes

Each software project that consumes packages to include the required dependencies will use the package manager and more package sources.

The package manager will download the individual packages from the sources and install them locally on the development machine or build server.

The developer flow will follow this general pattern:

1. Identify a required dependency in your codebase.
2. Find a component that satisfies the requirements for the project.
3. Search the package sources for a package offering a correct component version.
4. Install the package into the codebase and development machine.
5. Create the software implementation that uses the new components from the package.

The package manager tooling will help search and install the components in the packages.

How it's achieved varies for the different package types. Refer to the documentation of the package manager for instructions on consuming packages from feeds.

To get started, you'll need to specify the package source. Package managers will have a default source defined as the standard package feed for its type.

Alternative feeds will need to be configured to allow consuming the packages they offer.

## Upstream sources

Part of package management involves keeping track of the various sources.

It's possible to refer to multiple sources from a single software solution. However, when combining private and public sources, the order of resolution of the sources becomes essential.

One way to specify multiple package sources is by choosing a primary source and an upstream source.

The package manager will evaluate the primary source first and switch to the upstream source when the package isn't found there.

The upstream source might be one of the official public or private sources. The upstream source could refer to another upstream source, creating a chain of sources.

A typical scenario is to use a private package source referring to a public upstream source for one of the official feeds. It effectively enhances the packages in the upstream source with packages from the private feed, avoiding publishing private packages in a public feed.

A source with an upstream source defined may download and cache the requested packages if the source doesn't contain those packages themselves.

The source will include these downloaded packages and starts to act as a cache for the upstream source. It also offers the ability to keep track of any packages from the external upstream source.

An upstream source can be a way to avoid direct access of developers and build machines to external sources.

The private feed uses the upstream source as a proxy for the external source. It will be your feed manager and private source that have the communication to the outside. Only privileged roles can add upstream sources to a private feed.

See also [Upstream sources](#).

## Packages graph

A feed can have one or more upstream sources, which might be internal or external. Each of these can have additional upstream sources, creating a package graph of the source.

Such a graph can offer many possibilities for layering and indirection of origins of packages. It might fit well with multiple teams taking care of packages for frameworks and other base libraries.

The downside is that package graphs can become complex when not correctly understood or designed. It's essential to know how you can create a proper package graph.

See also [Constructing a complete package graph](#).

# Introduction to Azure Artifacts

Completed 100 XP

- 4 minutes

Previously you learned about packaging dependencies and the various packaging formats, feeds, sources, and package managers.

Now, you'll know more about package management, creating a feed, and publishing packages.

During this module, NuGet and Azure Artifacts are used as an example of a package format and a particular package feed and source type.

Microsoft Azure DevOps provides various features for application lifecycle management, including:

- Work item tracking.
- Source code repositories.
- Build and release pipelines.
- Artifact management.

The artifact management is called Azure Artifacts and was previously known as Package management. It offers public and private feeds for software packages of various types.

## Types of packages supported

Azure Artifacts currently supports feeds that can store five different package types:

- NuGet packages
- npm packages
- Maven
- Universal packages
- Python

Previously, we discussed the package types for NuGet, npm, Maven, and Python. Universal packages are an Azure Artifacts-specific package type. In essence, it's a versioned package containing multiple files and folders.

A single Azure Artifacts feed can contain any combination of such packages. You can connect to the feed using the package managers and the corresponding tooling for the package types. For Maven packages, this can also be the Gradle build tool.

## Selecting package sources

When creating your solution, you'll decide which packages you want to consume to offer the dependent components.

The next step is to determine the sources for these packages. The main choice is selecting public and private feeds or a combination of it.

Publicly available packages can usually be found in the public package sources. It would be nuget.org, npmjs, and pypi.org.

Your solution can select these sources if it only consumes packages available there.

Whenever your solution also has private packages that can't be available on public sources, you'll need to use a private feed.

In the previous module, you learned that public package sources could be upstream sources to private feeds.

Azure Artifacts allows its feeds to specify one or more upstream sources, public, or other private feeds.

# Publish packages

Completed 100 XP

- 4 minutes

As software is developed and components are written, you'll most likely produce components as dependencies packaged for reuse.

Discussed previously was guidance to find components that can be isolated into dependencies.

These components need to be managed and packaged. After that, they can be published to a feed, allowing others to consume the packages and use the components it contains.

## Creating a feed

The first step is to create a feed where the packages can be stored. In Azure Artifacts, you can create multiple feeds, which are always private.

During creation, you can specify the name, visibility and prepopulate the default public upstream sources for NuGet, npm, and Python packages.

## What are feeds in Azure Artifacts?

Most package management systems provide endpoints where you can request packages to install in your applications. Those endpoints are called feeds.

In Azure Artifacts, you can have multiple feeds in your projects, and you can make them available to only users authorized in your project or for your entire organization.

Each feed can contain any packages, even mixed types, but it's recommended that you create one feed per type you want to support in your organization, this way, it's clear what the feed contains.

Each feed can contain one or more upstream and can manage its security.

## Controlling access

The Azure Artifacts feed you created is always private and not available publically.

You need access to it by authenticating to Azure Artifacts with an account with access to Azure DevOps and a team project.

By default, a feed will be available to all registered users in Azure DevOps.

You can select it to be visible only to the team project where the feed is created.

Whichever option is chosen, you can change the permissions for a feed from the settings dialog.

## Push packages to a feed

Once you've authenticated to Azure DevOps, you can pull and push packages to the package feed, provided you have permission to do so.

Pushing packages is done with the tooling for the package manager. Each of the package managers and tooling has a different syntax for pushing.

To manually push a NuGet package, you would use the `NuGet.exe` command-line tool. For a package called `MyDemoPackage`, the command would resemble:

`CmdCopy`

```
nuget.exe push -Source {NuGet package source URL} -ApiKey YourKey  
YourPackage\YourPackage.nupkg
```

## Updating packages

Packages might need to be updated during their lifetime. Technically, updating a package is made by pushing a new version of the package to the feed.

The package feed manager manages to properly store the updated package with the existing packages in the feed.

### Note

Updating packages requires a versioning strategy.

# Package management with Azure Artifacts

Completed 100 XP

- 40 minutes

**Estimated time:** 40 minutes.

**Lab files:** none.

## Scenario

Azure Artifacts facilitate discovery, installation, and publishing NuGet, npm, and Maven packages in Azure DevOps. It's deeply integrated with other Azure DevOps features such as Build, making package management a seamless part of your existing workflows.

## Objectives

After completing this lab, you'll be able to:

- Create and connect to a feed.
- Create and publish a NuGet package.
- Import a NuGet package.
- Update a NuGet package.

# Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- Visual Studio 2022 Community Edition is available from the [Visual Studio Downloads page](#). Visual Studio 2022 installation should include **ASP.NET and web development**, **Azure development**, and **.NET Core cross-platform development** workloads.

# Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Working with Azure Artifacts.

# Explore package dependencies

## What is dependency management?

Completed 100 XP

- 1 minute

Before we can understand dependency management, we'll need first to get introduced to the concepts of dependencies.

### Dependencies in software

Modern software development involves complex projects and solutions.

Projects have dependencies on other projects, and solutions aren't single pieces of software.

The solutions and software built consist of multiple parts and components and are often reused.

As codebases are expanding and evolving, it needs to be componentized to be maintainable.

A team that is writing software won't write every piece of code by itself but use existing code written by other teams or companies and open-source code that is readily available.

Each component can have its maintainers, speed of change, and distribution, giving both the creators and consumers of the components autonomy.

A software engineer will need to identify the components that make up parts of the solution and decide whether to write the implementation or include an existing component.

The latter approach introduces a dependency on other components.

### Why is dependency management needed?

Software dependencies that are introduced in a project and solution must be appropriately declared and resolved.

You need to manage the overall composition of the project code and the included dependencies.

Without proper dependency management, it will be hard to keep the components in the solution controlled.

Management of dependencies allows a software engineer and team to be more efficient working with dependencies.

With all dependencies being managed, it's also possible to control the consumed dependencies, enabling governance and security scanning to use known vulnerabilities or exploits packages.

## Describe elements of a dependency management strategy

Completed 100 XP

- 1 minute

There are many aspects of a dependency management strategy.

- **Standardization** Managing dependencies benefit from a standardized way of declaring and resolving them in your codebase.  
Standardization allows a repeatable, predictable process and usage that can be automated as well.
- **Package formats and sources** The distribution of dependencies can be performed by a packaging method suited for your solution's dependency type.  
Each dependency is packaged using its usable format and stored in a centralized source.  
Your dependency management strategy should include the selection of package formats and corresponding sources where to store and retrieve packages.

- **Versioning** Just like your own code and components, the dependencies in your solution usually evolve.  
While your codebase grows and changes, you need to consider the changes in your dependencies as well.  
It requires a versioning mechanism for the dependencies to be selective of the version of a dependency you want to use.

## Identify dependencies

Completed 100 XP

- 1 minute

This process begins with identifying the dependencies in your codebase and determining which ones will be officially recognized.

Your software project and its associated solutions already use various dependencies. The use of libraries and frameworks authored by others is a widespread practice.

Your current codebase may contain internal dependencies that need to be officially acknowledged as such.

Consider a segment of code that encapsulates a specific business domain model. This code might be integrated as source code in your project and utilized across different projects and teams.

A thorough investigation of your codebase is necessary to pinpoint sections of code that should be classified as dependencies. This calls for modifying how you arrange your code and construct the solution. Eventually, it will contribute to the refinement of your components.

## Understand source and package componentization

Completed 100 XP

- 1 minute

Current development practices already have the notion of componentization.

There are two ways of componentization commonly used.

1. **Source componentization** The first way of componentization is focused on source code. It refers to splitting the source code in the codebase into separate parts and organizing it around the identified components.  
It works if the source code isn't shared outside of the project. Once the components need to be shared, it requires distributing the source code or the produced binary artifacts created from it.
2. **Package componentization** The second way uses packages. Distributing software components is performed utilizing packages as a formal way of wrapping and handling the components.  
A shift to packages adds characteristics needed for proper dependency management, like tracking and versioning packages in your solution.

See also [Collaborate more and build faster with packages](#).

## Decompose your system

Completed 100 XP

- 1 minute

You'll need to get better insights into your code and solution before you can change your codebase into separate components to prepare for finding dependencies that can be taken out of your system.

It allows you to decompose your system into individual components and dependencies. The goal is to reduce the size of your codebase and system, making it more efficient to build and manageable in the end.

You achieve it by removing specific components of your solution. These are going to be centralized, reused, and maintained independently.

You'll remove those components and externalize them from your solution at the expense of introducing dependencies on other components.

This process of finding and externalizing components is effectively creating dependencies.

It may require some refactoring, such as creating new solution artifacts for code organization or code changes to cater for the unchanged code to take a dependency on an (external) component.

You might need to introduce some code design patterns to isolate and include the componentized code.

Examples of patterns are abstraction by interfaces, dependency injection, and inversion of control.

Decomposing could also mean replacing your implementation of reusable code with an available open-source or commercial component.

## Scan your codebase for dependencies

Completed 100 XP

- 2 minutes

There are several ways to identify the dependencies in your codebase.

These include scanning your code for patterns and reuse and analyzing how the solution is composed of individual modules and components.

- **Duplicate code** When certain pieces of code appear in several places, it's a good indication that this code can be reused. Keep in mind that code duplication isn't necessarily a bad practice. However, if the code can be made available properly, it does have benefits over copying code and must manage that. The first step to isolate these pieces of duplicate code is to centralize them in the codebase and componentize them in the appropriate way for the type of code.
- **High cohesion and low coupling** A second approach is to find code that might define components in your solution. You'll look for code elements that have high cohesion and low coupling with other parts of code. It could be a specific object model with business logic or code related to its responsibility, such as a set of helper or utility codes or perhaps a basis for other code to be built upon.
- **Individual lifecycle** Related to high cohesion, you can look for parts of the code that have a similar lifecycle and can be deployed and released individually. If such code can be maintained by a team separate from the

codebase that it's currently in, it's an indication that it could be a component outside of the solution.

- **Stable parts** Some parts of your codebase might have a slow rate of change. That code is stable and isn't altered often. You can check your code repository to find the code with a low change frequency.
- **Independent code and components** Whenever code and components are independent and unrelated to other parts of the system, they can be isolated to a separate component and dependency.

You can use different kinds of tools to assist you in scanning and examining your codebase.

These range from tools that scan for duplicate code and draw solution dependency graphs to tools that compute metrics for coupling and cohesion.

# **Implement tools to track usage and flow**

## **Understand the inner loop**

Completed 100 XP

- 1 minute

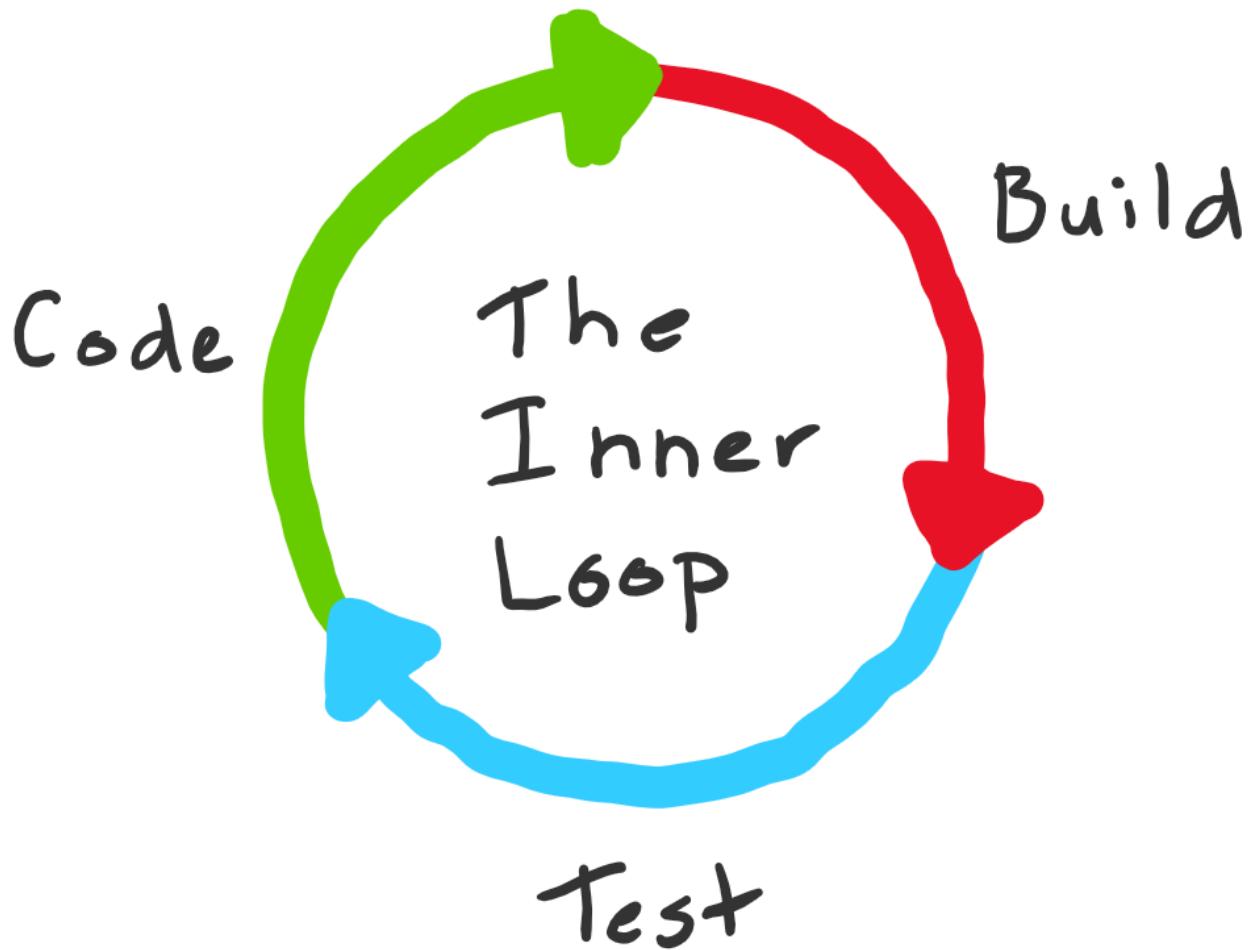
It isn't clear who coined the term "inner loop" in the context of software engineering, but within Microsoft, at least, the word seems to have stuck.

Many of the internal teams I work with see it as something they want to keep as short as possible - but what is the inner loop?

### **Definitions**

The easiest way to define the inner loop is the iterative process that a developer does when writing, building, and debugging code.

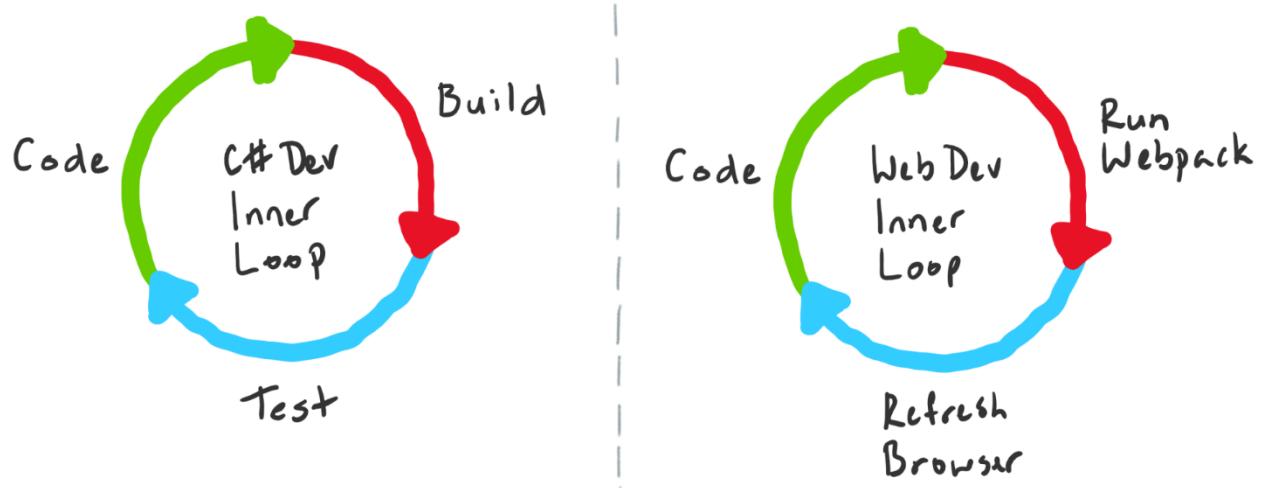
There are other things that a developer does. It's the right set of steps to be done repeatedly before sharing their work with their team or the rest of the world.



Exactly what goes into an individual developer's inner loop will depend significantly on the technologies they're working with, the tools used, and their preferences.

If I were working on a library, my inner loop would include coding, building, testing execution & debugging with regular commits to my local Git repository.

On the other hand, if I were doing web front-end work, I would probably be optimized around hacking on HTML & JavaScript, bundling, and refreshing the browser (followed by regular commits).



Most codebases comprise multiple-moving parts, so the definition of a developer's inner loop on any single codebase might alternate depending on what is being worked on.

## Understanding the loop

The steps within the inner loop can be grouped into three broad activity buckets - experimentation, feedback collection, and tax.

If we flick back to the library development scenario I mentioned, I said four steps and how to bucket them.

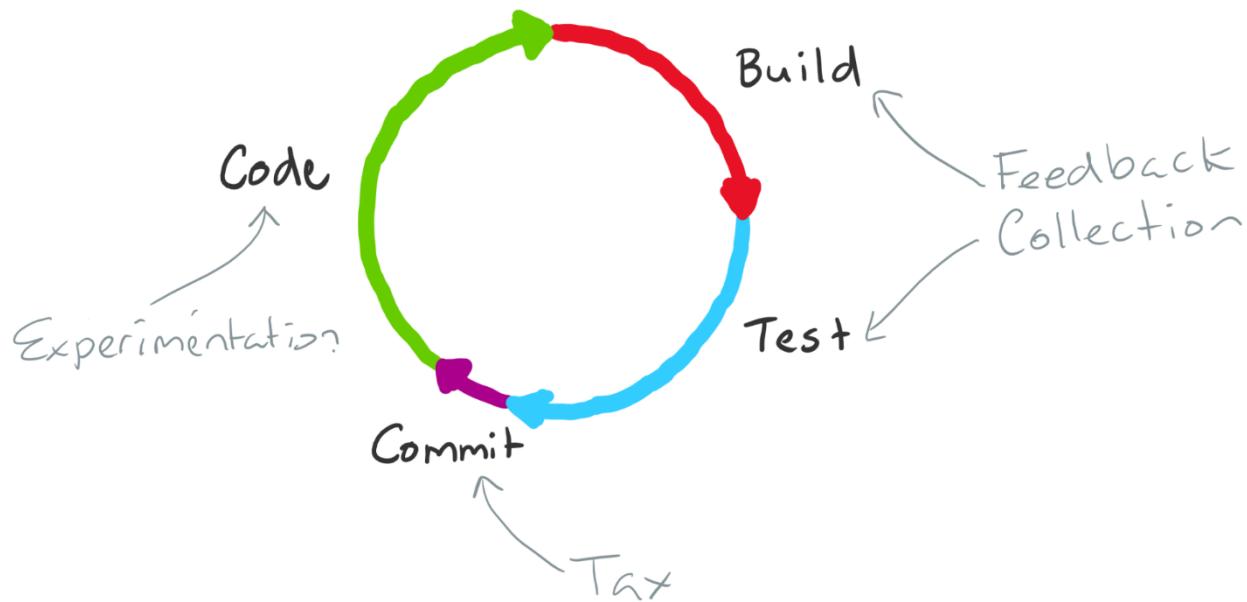
- Coding (Experimentation)
- Building (Feedback Collection)
- Testing / Debugging (Feedback Collection)
- Committing (Tax)

Of all the steps in the inner loop, coding is the only one that adds customer value.

Building and testing code is essential, but ultimately, we use them to give the developer feedback about their writing to see if it delivers sufficient value.

Putting committing code in the tax bucket is perhaps a bit harsh, but the purpose of the bucket is to call out those activities that neither add value nor provide feedback.

Tax is necessary to work. If it's unnecessary work, it's waste and should be eliminated.



## Loop optimization

Having categorized the steps within the loop, it's now possible to make some general statements:

- You want to execute the loop as fast as possible and for the total loop execution time to be proportional to the changes made.
- You want to minimize the time feedback collection takes but maximize the quality of the feedback that you get.
- You want to minimize the tax you pay by eliminating it where it's unnecessary to run through the loop (can you defer some operations until you commit, for example).
- As new code and more complexity is added to any codebase, the amount of outward pressure to increase the size of the inner loop also increases. More code means more tests, which means more execution time and slow execution of the inner loop.

Suppose you have ever worked on a large monolithic codebase. In that case, it's possible to get into a situation where even small changes require a disproportionate amount of time to execute the feedback collection steps of the inner loop. It's a problem, and you should fix it.

There are several things that a team can do to optimize the inner loop for larger codebases:

- Only build and test what was changed.
- Cache intermediate build results to speed up to complete builds.
- Break up the codebase into small units and share binaries.

How you tackle each one of those is probably a blog post.

At Microsoft, for some of our genuinely massive monolithic codebases.

We're investing heavily in #1 and #2, but #3 requires a special mention because it can be a double-edged sword and can have the opposite of the wished impact if done incorrectly.

## Tangled loops

To understand the problem, we need to look beyond the inner loop. Let us say that our monolithic codebase has an application-specific framework that does much heavy lifting.

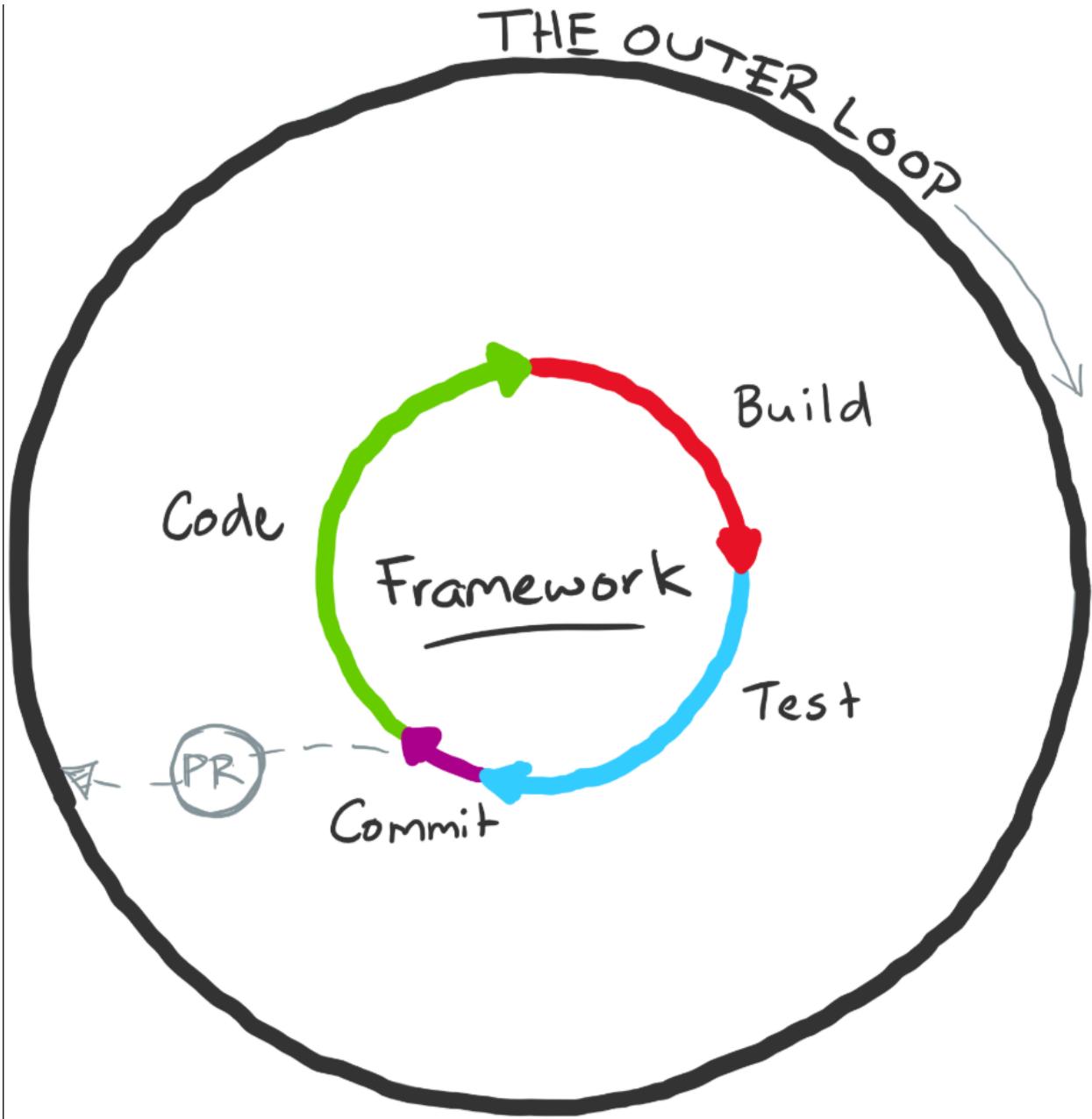
It would be tempting to extract that framework into a set of packages.

To do this, you would pull that code into a separate repository (optional, but this is generally how it's done), then set up a different CI/CD pipeline that builds and publishes the package.

A different pull-request process would also front this separate build and release pipeline to inspect changes before the code is published.

When someone needs to change this framework code, they clone down the repository, make changes (a separate inner loop), and submit a PR that transitions the workflow from the inner loop to the outer loop.

The framework package would then be available to be pulled into dependent applications (in this case, the monolith).



Initially, things might work out well. However, at some point in the future, you'll likely want to develop a new feature in the application that requires extensive new capabilities to be added to the framework.

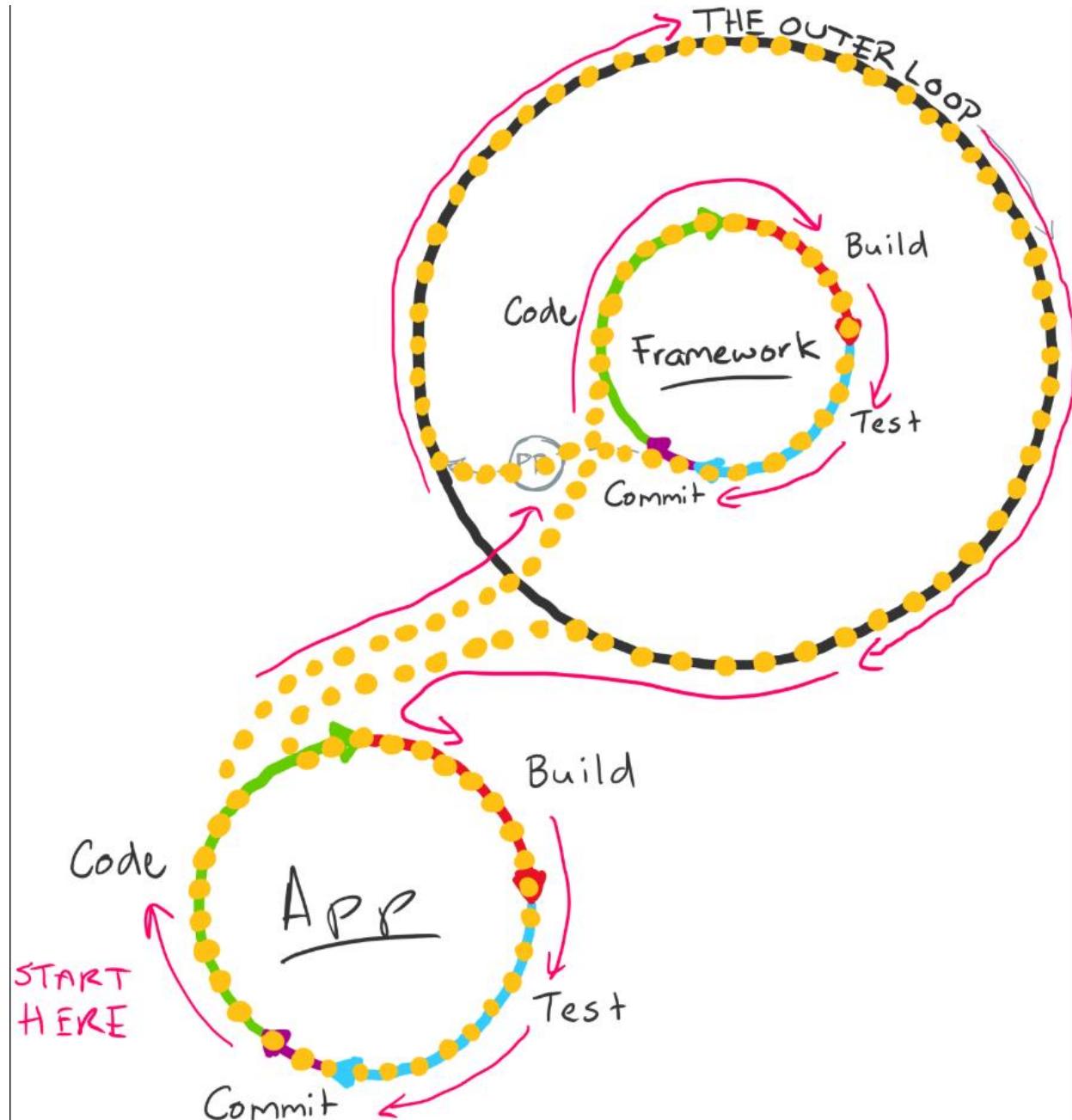
It's where teams that have broken up their codebases in suboptimal ways will start to feel pain.

If you have to coevolve code in two separate repositories where a binary/library dependency is present, you'll experience some friction.

In loop terms, the original codebase's inner loop now (temporarily at least) includes the outer loop of the previously broken-out framework code.

Outer loops include tax, including code reviews, scanning passes, binary signing, release pipelines, and approvals.

You don't want to pay that every time you've added a method to a class in the framework and now want to use it in your application.



What generally ends up happening next is a series of local hacks by the developer to stitch the inner loops together so they can move forward efficiently - but it gets messy quickly. You must pay that outer loop tax at some point.

It isn't to say that breaking code into separate packages is inherently bad - it can work brilliantly; you need to make those incisions carefully.

## Closing thoughts

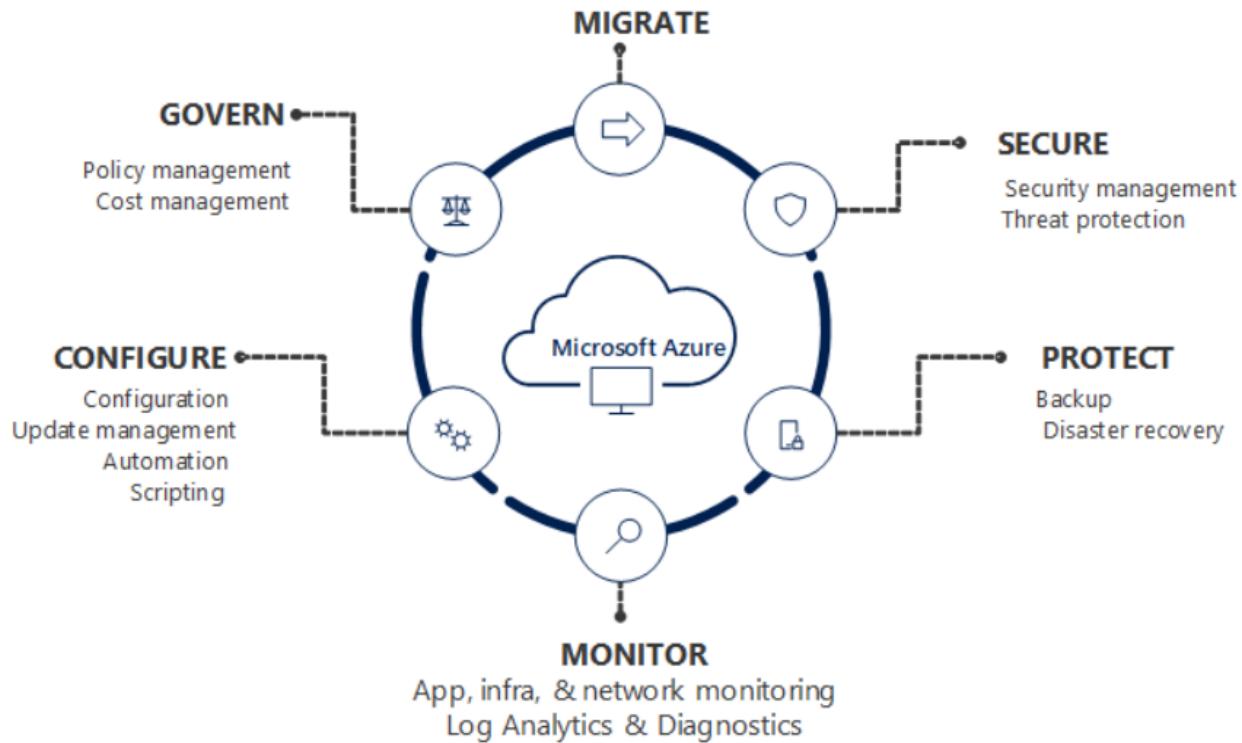
There's no silver bullet solution to ensure that your inner loop doesn't start slowing down, but it's essential to understand when it starts happening, what the cause is, and work to address it.

Decisions such as how you build, test, and debug the architecture will all impact developers' productivity. Improving one aspect will often cause issues in another.

# Introduction to continuous monitoring

Completed 100 XP

- 4 minutes



Continuous monitoring refers to the process and technology required to incorporate monitoring across each DevOps and IT operations lifecycles phase.

It helps to continuously ensure your application's health, performance, reliability, and infrastructure as it moves from development to production.

Continuous monitoring builds on the concepts of Continuous Integration and Continuous Deployment (CI/CD), which help you develop and deliver software faster and more reliably to provide continuous value to your users.

[Azure Monitor](#) is the unified monitoring solution in Azure that provides full-stack observability across applications and infrastructure in the cloud and on-premises.

It works seamlessly with [Visual Studio and Visual Studio Code](#) during development and testing and integrates with [Azure DevOps](#) for release management and work item management during deployment and operations.

It even integrates across your ITSM and SIEM tools to help track issues and incidents within your existing IT processes.

This article describes specific steps for using Azure Monitor to enable continuous monitoring throughout your workflows.

It includes links to other documentation that provides details on implementing different features.

## Enable monitoring for all your applications

To gain observability across your entire environment, you need to enable monitoring on all your web applications and services.

It will allow you to visualize end-to-end transactions and connections across all the components easily.

- Azure DevOps Projects gives you a simplified experience with your existing code and Git repository or choose from one of the sample applications to create a Continuous Integration (CI) and Continuous Delivery (CD) pipeline to Azure.
- Continuous monitoring in your DevOps release pipeline allows you to gate or roll back your deployment based on monitoring data.
- Status Monitor allows you to instrument a live .NET app on Windows with Azure Application Insights without modifying or redeploying your code.
- If you have access to the code for your application, then enable complete monitoring with Application Insights by installing the Azure Monitor Application Insights SDK for .NET, Java, Node.js, or any other programming language. It lets you specify custom events, metrics, or page views relevant to your application and business.

## Enable monitoring for your entire infrastructure

Applications are only as reliable as their underlying infrastructure.

Monitoring enabled across your entire infrastructure will help you achieve full observability and make discovering a potential root cause easier when something fails.

Azure Monitor helps you track the health and performance of your entire hybrid infrastructure, including resources such as VMs, containers, storage, and network.

- You automatically get platform metrics, activity logs, and diagnostics logs from most of your Azure resources with no configuration.
- Enable deeper monitoring for VMs with Azure Monitor.
- Enable deeper monitoring for AKS clusters with Azure Monitor for containers.
- Add monitoring solutions for different applications and services in your environment.

**Infrastructure as code** manages infrastructure in a descriptive model, using the same versioning as DevOps teams use for source code.

It adds reliability and scalability to your environment and allows you to apply similar processes to manage your applications.

- Use [Resource Manager templates](#) to enable monitoring and configure alerts over a large set of resources.
- Use [Azure Policy](#) to enforce different rules over your resources. It ensures those resources comply with your corporate standards and service level agreements.

## Combine resources in Azure Resource Groups

Today, a typical application on Azure includes multiple resources such as VMs and App Services or microservices hosted on Cloud Services, AKS clusters, or Service Fabric.

These applications frequently use dependencies like Event Hubs, Storage, SQL, and Service Bus.

- Combine resources in Azure Resource Groups to get complete visibility of all the resources that make up your different applications. [Azure Monitor for Resource Groups](#) provides a simple way to keep track of the health and performance of your entire full-stack application and enables drilling down into respective components for any investigations or debugging.

## Ensure quality through continuous deployment

[Continuous Integration / Continuous Deployment](#) allows you to automatically integrate and deploy code changes to your application based on automated testing results.

It streamlines the deployment process and ensures the quality of any changes before they move into production.

- Use [Azure Pipelines](#) to implement Continuous Deployment and automate your entire process from code commit to production based on your CI/CD tests.
- Use Quality Gates to integrate monitoring into your pre-deployment or post-deployment. It ensures that you meet the key health/performance metrics (KPIs) as your applications move from dev to production. Any differences in the infrastructure environment or scale aren't negatively impacting your KPIs.

- Maintain separate monitoring instances between your different deployment environments, such as Dev, Test, Canary, and Prod. It ensures that collected data is relevant across the associated applications and infrastructure. If you need to correlate data across environments, use [multi-resource charts in Metrics Explorer](#) or [create cross-resource queries in Log Analytics](#).

## Create actionable alerts with actions

A critical monitoring aspect is proactively notifying administrators of current and predicted issues.

- Create [alerts in Azure Monitor](#) based on logs and metrics to identify predictable failure states. It would be best if you had a goal of making all alerts actionable, meaning that they represent actual critical conditions and seek to reduce false positives. Use [dynamic thresholds](#) to automatically calculate baselines on metric data rather than defining your static thresholds.
- Define actions for alerts to use the most effective means of notifying your administrators. Available [actions for notification](#) are SMS, e-mails, push notifications or voice calls.
- Use more advanced actions to [connect to your ITSM tool](#) or other alert management systems through [webhooks](#).
- Remediate situations identified in alerts with [Azure Automation runbooks](#) or [Logic Apps](#) that can be launched from an alert using webhooks.
- Use [autoscaling](#) to dynamically increase and decrease your compute resources based on collected metrics.

## Prepare dashboards and workbooks

Ensuring that your development and operations have access to the same telemetry and tools allows them to view patterns across your entire environment and minimize your Mean Time To Detect (MTTD) and Mean Time To Restore (MTTR).

- Prepare [custom dashboards](#) based on standard metrics and logs for the different roles in your organization. Dashboards can combine data from all Azure resources.
- Prepare [Workbooks](#) to ensure knowledge sharing between development and operations. It could be prepared as dynamic reports with metric charts and log queries or as troubleshooting guides designed by developers to help customer support or operations handle fundamental problems.

## Continuously optimize

Monitoring is one of the fundamental aspects of the popular Build-Measure-Learn philosophy, which recommends continuously tracking your KPIs and user behavior metrics and optimizing them through planning iterations.

Azure Monitor helps you collect metrics and logs relevant to your business and add new data points in the following deployment.

- Use tools in Application Insights to [track end-user behavior and engagement](#).
- Use [Impact Analysis](#) to help you prioritize which areas to focus on to drive to important KPIs.

## Explore Azure Monitor and Log Analytics

Completed 100 XP

- 6 minutes

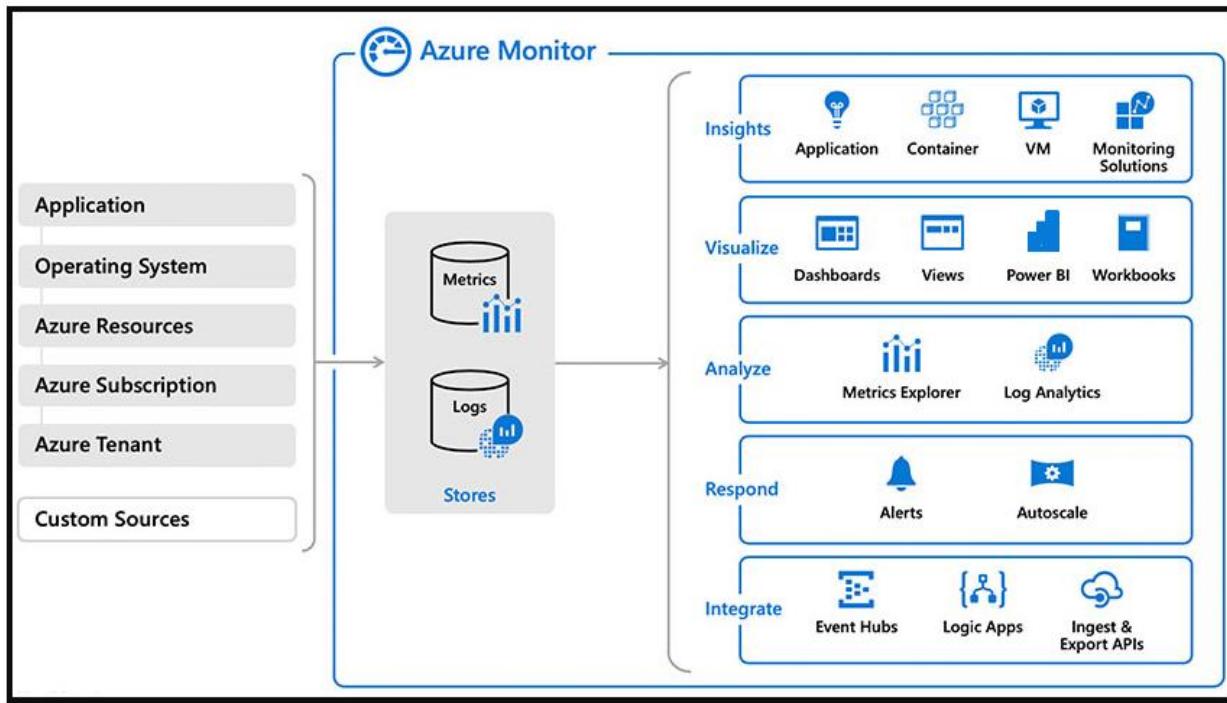
When you run at a cloud scale, you need intelligent logging and monitoring tools that scale to your needs and provide real-time insight into your data.

Azure Monitor is Microsoft's native cloud monitoring solution. Azure Monitor collects monitoring telemetry from different kinds of on-premises and Azure sources.

Azure Monitor provides Management tools, such as those in Azure Security Center and Azure Automation, enabling ingestion of custom log data to Azure.

The service aggregates and stores this telemetry in a log data store optimized for cost and performance.

With Azure Monitor, you can analyze data, set up alerts, and get end-to-end views of your applications. And use machine-learning-driven insights to identify and resolve problems quickly.



In this tutorial, we'll focus on the Log Analytics part of Azure Monitor. We'll learn how to:

- Set up Log Analytics workspace.
- Connect virtual machines to a log analytics workspace.
- Configure Log Analytics workspace to collect custom performance counters.
- Analyze the telemetry using Kusto Query Language.

## Getting started

1. You'll need a resource group with one or more virtual machines that you have access to RDP to follow along.
2. Log into [Azure Shell](#). Execute the command below. It will create a new resource group and log analytics workspace. Take a record of the workspaceid of the log analytics workspace as we'll be using it again.

PowerShellCopy

```
$ResourceGroup = "azwe-rg-devtest-logs-001"
$WorkspaceName = "azwe-devtest-logs-01"
$Location = "westeurope"

# List of solutions to enable
$Solutions = "CapacityPerformance", "LogManagement", "ChangeTracking",
"ProcessInvestigator"
```

```

# Create the resource group if needed
try {
    Get-AzResourceGroup -Name $ResourceGroup -ErrorAction Stop
} catch {
    New-AzResourceGroup -Name $ResourceGroup -Location $Location
}

# Create the workspace
New-AzOperationalInsightsWorkspace -Location $Location -Name
$WorkspaceName -ResourceGroupName $ResourceGroup

# List all solutions and their installation status
Get-AzOperationalInsightsIntelligencePacks -ResourceGroupName
$ResourceGroup -WorkspaceName $WorkspaceName

# Add solutions
foreach ($solution in $Solutions) {
    Set-AzOperationalInsightsIntelligencePack -ResourceGroupName
    $ResourceGroup -WorkspaceName $WorkspaceName -IntelligencePackName
    $solution -Enabled $true
}

# List enabled solutions
(Get-AzOperationalInsightsIntelligencePacks -ResourceGroupName
$ResourceGroup -WorkspaceName $WorkspaceName).Where({($_.enabled -eq
$true)})

# Enable IIS Log Collection using the agent
Enable-AzOperationalInsightsIISLogCollection -ResourceGroupName
$ResourceGroup -WorkspaceName $WorkspaceName

# Windows Event
New-AzOperationalInsightsWindowsEventDataSource -ResourceGroupName
$ResourceGroup -WorkspaceName $WorkspaceName -EventLogName "Application"
-CollectErrors -CollectWarnings -Name "Example Application Event Log"

```

3. Retrieve the Log Analytics workspace secure key.

PowerShellCopy

```

Get-AzOperationalInsightsWorkspaceSharedKey ` 
    -ResourceGroupName azwe-rg-devtest-logs-001 ` 
    -Name azwe-devtest-logs-01

```

4. Map existing virtual machines with the Log Analytics workspace. The following query uses the workspaceid and workspace-secret key of the log analytics workspace to install the Microsoft Enterprise Cloud Monitoring extension onto an existing VM.

PowerShellCopy

```
$PublicSettings = @{"workspaceId" = "<myWorkspaceId>"}  
$ProtectedSettings = @{"workspaceKey" = "<myWorkspaceKey>"}  
  
Set-AzVMExtension -ExtensionName "Microsoft.EnterpriseCloud.Monitoring"  
    -ResourceGroupName "azwe-rg-devtest-logs-001"  
    -VMName "azsu-d-sql01-01"  
    -Publisher "Microsoft.EnterpriseCloud.Monitoring"  
    -ExtensionType "MicrosoftMonitoringAgent"  
    -TypeHandlerVersion 1.0  
    -Settings $PublicSettings  
    -ProtectedSettings $ProtectedSettings  
    -Location westeurope
```

5. Run the script to configure the below-listed performance counters to be collected from the virtual machine.

PowerShellCopy

```
#Login-AzureRmAccount  
  
#Instance  
#####  
$InstanceNameAll = "*"  
$InstanceNameTotal = '_Total'  
#Objects  
#####  
$ObjectCache = "Cache"  
$ObjectLogicalDisk = "LogicalDisk"  
$ObjectMemory = "Memory"  
$ObjectNetworkAdapter = "Network Adapter"  
$ObjectNetworkInterface = "Network Interface"  
$ObjectPagingFile = "Paging File"  
$ObjectProcess = "Process"  
$ObjectProcessorInformation = "Processor Information"  
$ObjectProcessor = "Processor"  
  
$ObjectSQLAgentAlerts = "SQLAgent:Alerts"  
$ObjectSQLAgentJobs = "SQLAgent:Jobs"  
$ObjectSQLAgentStatistics = "SQLAgent:Statistics"  
  
$ObjectSqlServerAccessMethods = "SQLServer:Access Methods"  
$ObjectSqlServerExecStatistics = "SQLServer:Exec Statistics"  
$ObjectSqlServerLocks = "SQLServer:Locks"  
$ObjectSqlServerSQLErrors = "SQLServer:SQL Errors"  
  
$ObjectSystem = "System"  
  
#Counters  
#####
```

```

$CounterCache = "Copy Read Hits %"

$CounterLogicalDisk =
    "% Free Space"
    , "Avg. Disk sec/Read"
    , "Avg. Disk sec/Transfer"
    , "Avg. Disk sec/Write"
    , "Current Disk Queue Length"
    , "Disk Read Bytes/sec"
    , "Disk Reads/sec"
    , "Disk Transfers/sec"
    , "Disk Writes/sec"

$CounterMemory =
    "% Committed Bytes In Use"
    , "Available MBytes"
    , "Page Faults/sec"
    , "Pages Input/sec"
    , "Pages Output/sec"
    , "Pool Nonpaged Bytes"

$CounterNetworkAdapter =
    "Bytes Received/sec"
    , "Bytes Sent/sec"

$CounterNetworkInterface = "Bytes Total/sec"

$CounterPagingFile =
    "% Usage"
    , "% Usage Peak"

$CounterProcess = "% Processor Time"

$CounterProcessorInformation =
    "% Interrupt Time"
    , "Interrupts/sec"

$CounterProcessor = "% Processor Time"
$CounterProcessorTotal = "% Processor Time"

$CounterSQLAgentAlerts = "Activated alerts"
$CounterSQLAgentJobs = "Failed jobs"
$CounterSQLAgentStatistics = "SQL Server restarted"
$CounterSQLServerAccessMethods = "Table Lock Escalations/sec"
$CounterSQLServerExecStatistics = "Distributed Query"
$CounterSQLServerLocks = "Number of Deadlocks/sec"
$CounterSQLServerSQLErrors = "Errors/sec"

$CounterSystem = "Processor Queue Length"

#####
$global:number = 1 #Name parameter needs to be unique that why we will
use number ++ in function
#####

```

```

function AddPerfCounters ($PerfObject, $PerfCounters, $Instance)
{
    ForEach ($Counter in $PerfCounters)
    {
        New-AzOperationalInsightsWindowsPerformanceCounterDataSource ` 
            -ResourceGroupName 'azwe-rg-devtest-logs-001' ` 
            -WorkspaceName 'azwe-devtest-logs-01' ` 
            -ObjectName $PerfObject ` 
            -InstanceName $Instance ` 
            -CounterName $Counter ` 
            -IntervalSeconds 10 ` 
            -Name "Windows Performance Counter $global:number"
        $global:number ++
    }
}

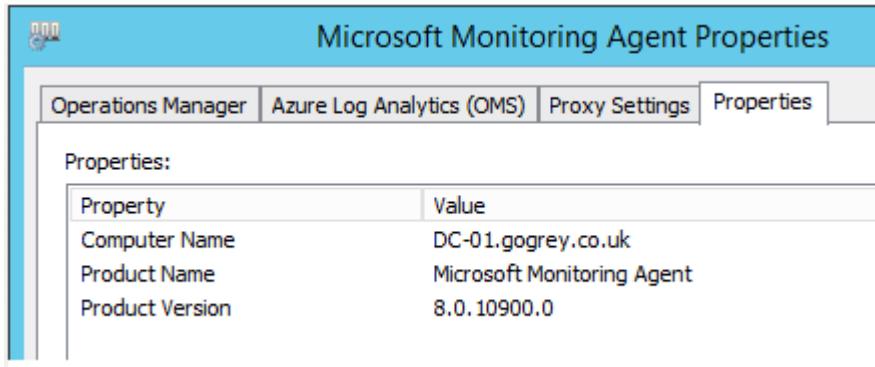
AddPerfCounters -PerfObject $ObjectLogicalDisk -PerfCounter
$CounterLogicalDisk -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectNetworkAdapter -PerfCounter
$CounterNetworkAdapter -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectNetworkInterface -PerfCounter
$CounterNetworkInterface -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectPagingFile -PerfCounter
$CounterPagingFile -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectProcess -PerfCounter $CounterProcess - 
Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectProcessorInformation -PerfCounter
$CounterProcessorInformation -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectProcessor -PerfCounter
$CounterProcessor -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectProcessor -PerfCounter
$CounterProcessorTotal -Instance $InstanceNameTotal
AddPerfCounters -PerfObject $ObjectSQLAgentAlerts -PerfCounter
$CounterSQLAgentAlerts -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectSQLAgentJobs -PerfCounter
$CounterSQLAgentJobs -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectSQLAgentStatistics -PerfCounter
$CounterSQLAgentStatistics -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectSQLServerAccessMethods -PerfCounter
$CounterSQLServerAccessMethods -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectSQLServerExecStatistics -PerfCounter
$CounterSQLServerExecStatistics -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectSQLServerLocks -PerfCounter
$CounterSQLServerLocks -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectSQLServerSQLErrors -PerfCounter
$CounterSQLServerSQLErrors -Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectSystem -PerfCounter $CounterSystem - 
Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectMemory -PerfCounter $CounterMemory - 
Instance $InstanceNameAll
AddPerfCounters -PerfObject $ObjectCache -PerfCounter $CounterCache - 
Instance $InstanceNameAll

```

6. To generate some interesting performance statistics. Download the [HeavyLoad utility](#) (a free load testing utility) and run it on the virtual machine to simulate high CPU, Memory, and IOPS consumption.

## How it works

1. Log Analytics works by running the Microsoft Monitoring Agent service on the machine. The service locally captures and buffers the events and pushes them securely out to the Log Analytics workspace in Azure.
2. Log into the virtual machine, navigate to the C:\Program Files\Microsoft Monitoring Agent\MMA, and open the control panel. It will show you the details of the log analytics workspace connected. You also can add multiple log analytics workspaces to publish the log data into various workspaces.



## Summary

So far, we've created a log analytics workspace in a resource group.

The log analytics workspace has been configured to collect performance counters, event logs, and IIS Logs.

The Microsoft Enterprise cloud monitoring extension has mapped a virtual machine to the log analytics workspace.

HeavyLoad has been used to simulate high CPU, memory, and IOPS on the virtual machine.

# Examine Kusto Query Language (KQL)

Completed 100 XP

- 5 minutes

Kusto is the primary way to query Log Analytics. It provides both a query language and a set of control commands.

Kusto can be used directly within Azure Data Explorer.

Azure Data Studio also offers a Kusto query experience and supports the creation of Jupiter-style notebooks for Kusto queries.

See [Getting Started with Kusto Queries](#).

## Walkthrough

Note: This walkthrough continues the previous lesson on Azure Log Analytics, and the walkthrough started within it.

1. Log in to the [Azure portal](#) and navigate to the log analytics workspace. From the left blade in the log analytics workspace, click Logs. It will open the Logs window, ready for you to start exploring all the data points captured into the workspace.
2. We'll need to use the Kusto Query Language to query the logs. Run the following query to list the last heartbeat of each machine connected to the log analytics workspace.

C#Copy

```
// Last heartbeat of each computer
// Show the last heartbeat sent by each computer
Heartbeat
| summarize arg_max(TimeGenerated, *) by Computer
```

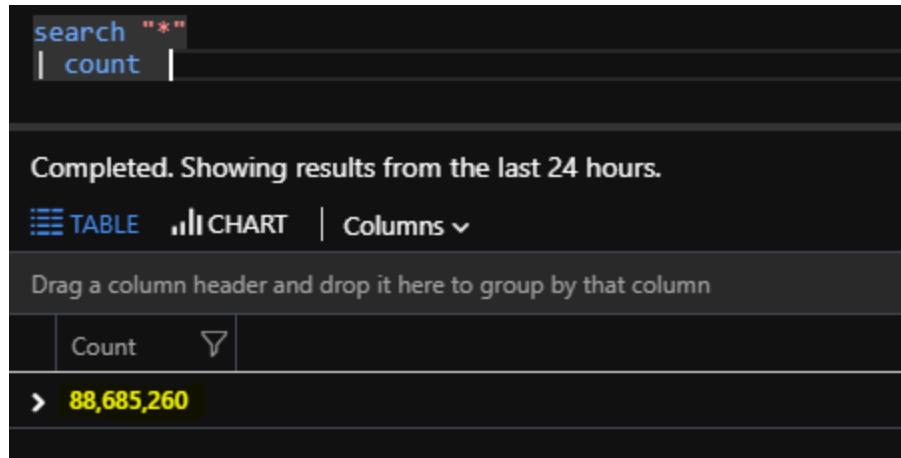
3. Show a list of all distinct counters being captured.

C#Copy

```
// What data is being collected?
// List the collected performance counters and object types (Process,
Memory, Processor.)
```

```
Perf  
| summarize by ObjectName, CounterName
```

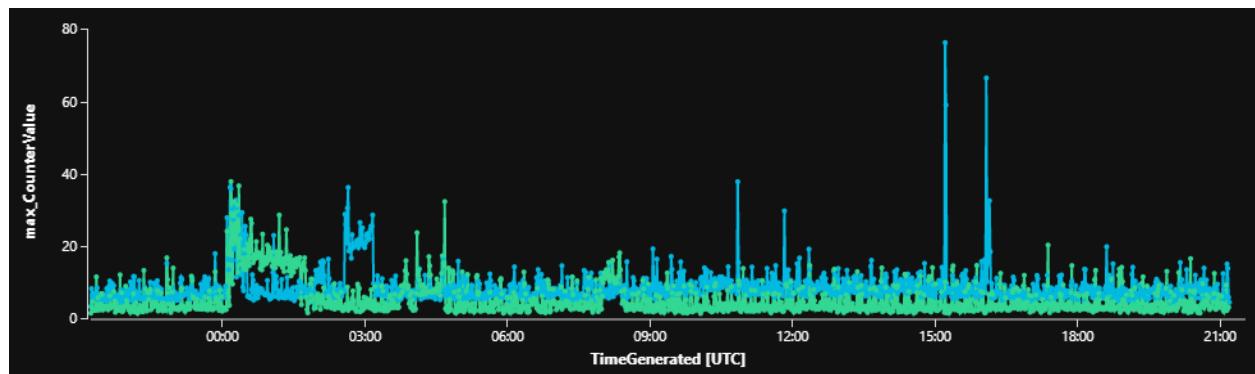
4. Show a count of the data points collected in the last 24 hours. The result shows that we have 88M data points. We can query against them in near real-time to analyze and correlate insights.



5. Run the following query to generate the max CPU Utilization trend over the last 24 hours, aggregated at a granularity of 1 min. Render the data as a time chart.

C#Copy

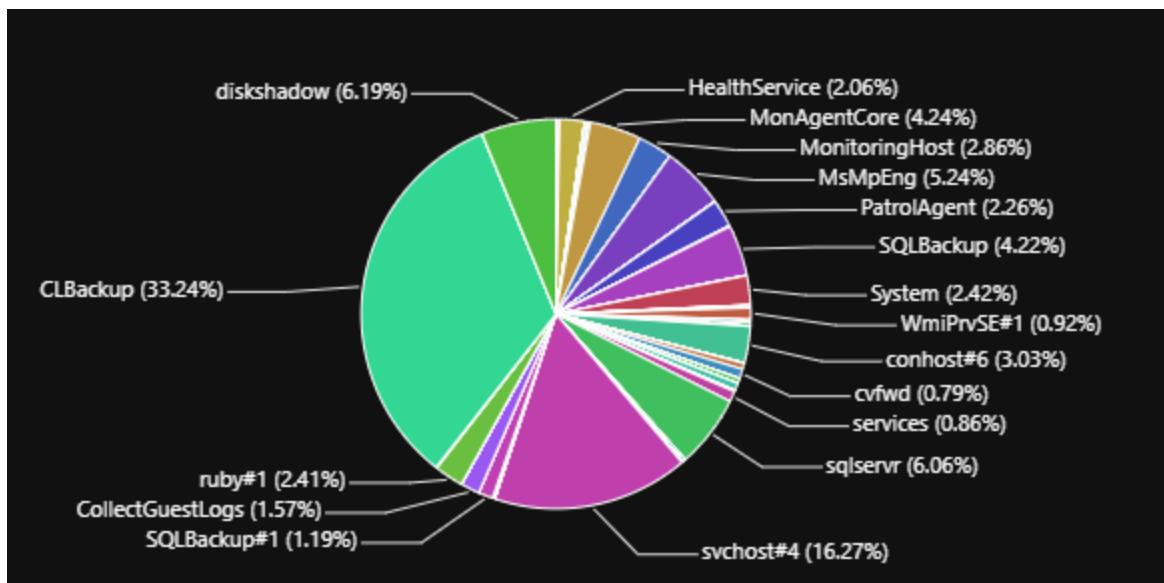
```
Perf  
| where ObjectName == "Processor" and InstanceName == "_Total"  
| summarize max(CounterValue) by Computer, bin(TimeGenerated, 1m)  
| render timechart
```



6. Run the following query to see all the processes running on that machine contributing to the CPU Utilization. Render the data in a pie chart.

C#Copy

```
Perf  
| where ObjectName contains "process"  
    and InstanceName !in ("_Total", "Idle")  
    and CounterName == "% Processor Time"  
| summarize avg(CounterValue) by InstanceName, CounterName,  
bin(TimeGenerated, 1m)  
| render piechart
```



## There's more

This unit has introduced the basic concepts of Log Analytics and how to get started with the basics.

We've only scratched the surface of what is possible with Log Analytics.

We would encourage you to try out the advanced tutorials available for Log Analytics on [Microsoft Docs](#).

## Explore Application Insights

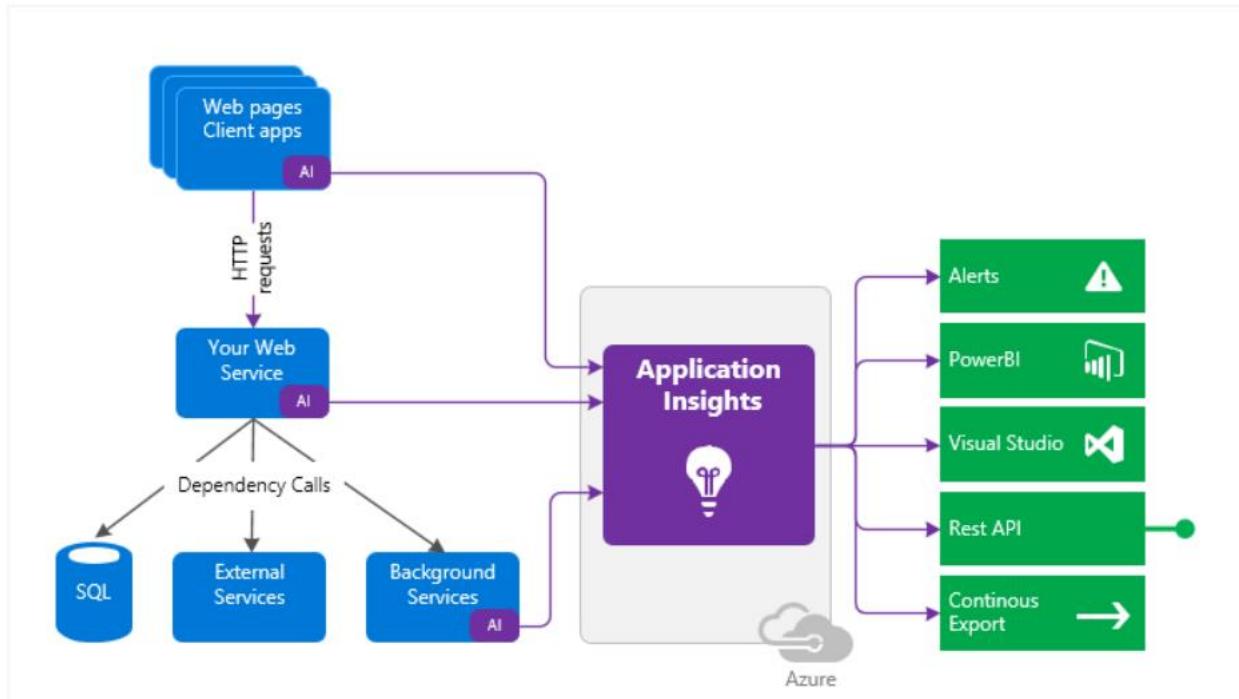
Completed 100 XP

- 5 minutes

You install a small instrumentation package in your application and set up an Application Insights resource in the Microsoft Azure portal.

The instrumentation monitors your app and sends telemetry data to the portal. (The application can run anywhere - it doesn't have to be hosted in Azure.)

You can instrument the web service application, background components, and JavaScript in the web pages.



Also, you can pull in telemetry from the host environments such as performance counters, Azure diagnostics, or Docker logs.

You can also set up web tests periodically, sending synthetic requests to your web service.

All these telemetry streams are integrated into the Azure portal, where you can apply powerful analytic and search tools to the raw data.

## What's the overhead?

The impact on your app's performance is minimal. Tracking calls are non-blocking and are batched and sent in a separate thread.

## What do Application Insights monitor?

Application Insights is aimed at the development team to help you understand how your app is doing and being used. It monitors:

- Request rates, response times, and failure rates - Find out which pages are most popular, at what times of day, and where your users are. See which pages do best. If your response times and failure rates increase with more requests, perhaps you have a resourcing problem.
- Dependency rates, response times, and failure rates - Find out whether external services are slowing you down.
- Exceptions - Analyze the aggregated statistics, pick specific instances, and drill into the stack trace and related requests. Both server and browser exceptions are reported.
- Pageviews and load performance - reported by your users' browsers.
- AJAX calls from web pages - rates, response times, and failure rates.
- User and session count.
- Performance counters from your Windows or Linux server machines include CPU, memory, and network usage.
- Host diagnostics from Docker or Azure.
- Diagnostic trace logs from your app - so you can correlate trace events with requests.
- Custom events and metrics that you write yourself in the client or server code to track business events such as items sold or games won.

## Where do I see my telemetry?

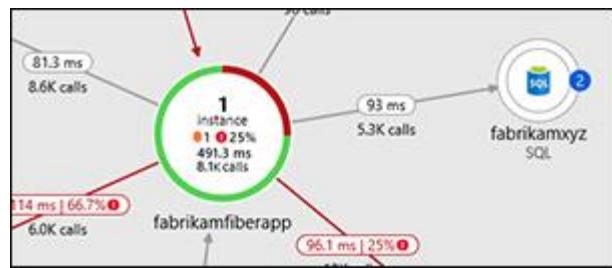
There are plenty of ways to explore your data. Check out this article for more information - [Smart detection and manual alerts](#).

Automatic alerts adapt to your app's usual patterns of telemetry and trigger when there's something outside the usual pattern. You can also [set alerts](#) on levels of custom or standard metrics.

| Abnormal rise in failed request rate in app "fabrikamprod"                 |                              |                         |
|--|------------------------------|-------------------------|
| Analysis time (UTC)  | Detected failed request rate | Normal rate (8 minutes) |
| 03/27/2016, 07:56 - 07:58  | 93.8% (45/48)                | 1%                      |
| 78% of the failed requests affected 1 user and have these characteristics: |                              |                         |
| Response code:   | 500 - Internal server error  |                         |
| Operation name:  | POST Customers/Create        |                         |

## Application map

The components of your app, with key metrics and alerts.



## Profiler

Inspect the execution profiles of sampled requests.



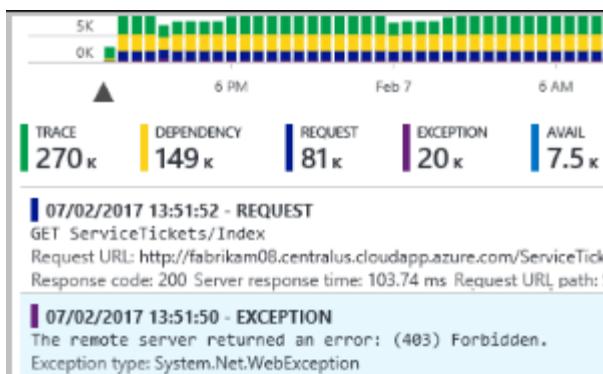
## Usage analysis

Analyze user segmentation and retention.

|                 | Users | Last returned after |       |       |       |     |
|-----------------|-------|---------------------|-------|-------|-------|-----|
|                 |       | <1w                 | +1w   | +2w   | +3w   | +4w |
| Overall         | 29    | 82.5%               | 23.3% | 20%   | 23.6% | 50% |
| Used on:        |       |                     |       |       |       |     |
| Apr 2 - Apr 8   | 2     | 100%                | 50%   | 50%   | 50%   | 50% |
| Apr 9 - Apr 15  | 5     | 100%                | 20%   | 20%   | 20%   |     |
| Apr 16 - Apr 22 | 13    | 76.0%               | 30.8% | 15.4% |       |     |
| Apr 23 - Apr 29 | 10    | 90%                 | 10%   |       |       |     |
| Apr 30 - now    | 8     | 100%                |       |       |       |     |

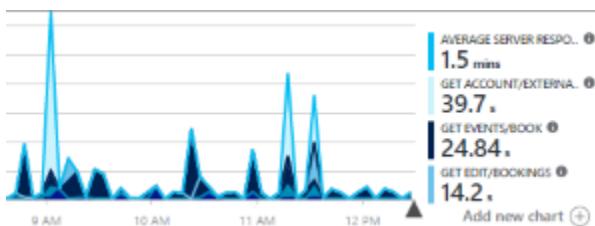
**Diagnostic search, for instance, data.**

Search and filter events such as requests, exceptions, dependency calls, log traces, and page views.



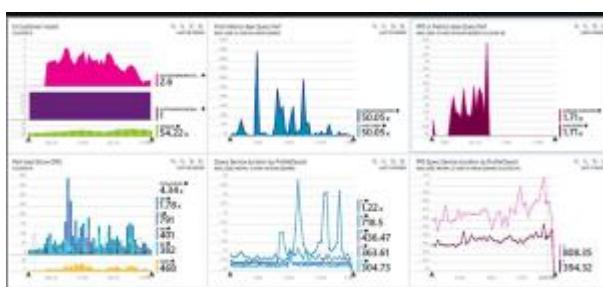
## Metrics Explorer for aggregated data

Explore, filter, and segment aggregated data such as rates of requests, failures, exceptions, response times, and page load times.



# Dashboards

Mash up data from multiple resources and share it with others. Great for multi-component applications and continuous display in the team room.



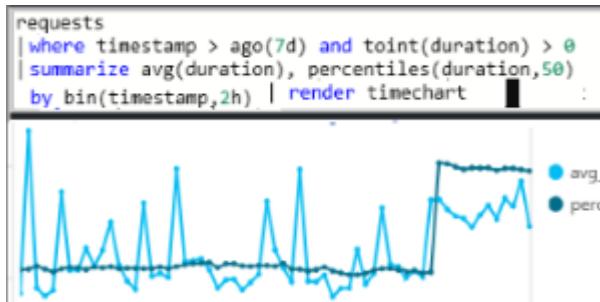
## Live Metrics Stream

When you deploy a new build, watch these near-real-time performance indicators to ensure everything works as expected.



## Analytics

Answer challenging questions about your app's performance and usage by using this powerful query language.



## Visual Studio

See performance data in the code. Go to code from stack traces.

Exception Details    Related Items

**Stack Trace**

```

    ▲ System.OperationCanceledException : no such op
      at Application2.Controllers.HomeController.About()
      at lambda_method
      at System.Web.Mvc.ActionMethodDispatcher.Execute
  
```

0 references | 1 exception

```

public ActionResult About()
{
    ViewBag.Message = "Your application desc
    if (DateTime.Now.IsDaylightSavingTime())
  
```

## Snapshot debugger

Debug snapshots sampled from live operations, with parameter values.

```

    /// <param name="visitorId"></param>
    /// <param name="pictureType">&lt;!--&lt;/param>
    /// <returns>Visitor</returns>
    [WebApiOutputCacheAttribute()]
    [Route("{visitorId:int:min(1)}")]
    [Route("~/oauth/api/visitors/{visitorId}")]
    public async Task<Visitor> Get(
    {
        var visitor = await _visitorRe
        var securityCode = "";
        if (visitor.SecurityCode != nu
        {
            // Expected format: BUILDING
            var parts = visitor.SecurityCode.Split('.');
            securityCode = St
        }
    }
  
```

**Exception Thrown**  
**System.IndexOutOfRangeException**: 'Index was outside the bounds of the array.'  
[View Details](#) | [Copy Details](#)  
Exception Settings  
 Break when this exception type is thrown  
 Except when thrown from:  
 MyCompany.Visitors.Web.dll  
[Open Exception Settings](#) | [Edit Conditions](#)

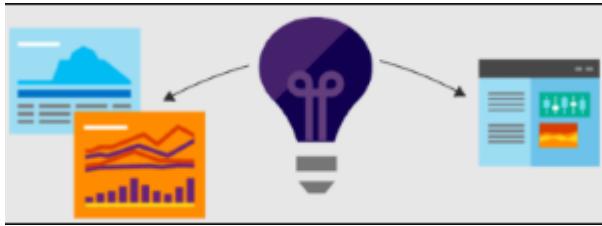
## Power BI

Integrate usage metrics with other business intelligence.



## REST API

Write code to run queries over your metrics and raw data.



## Continuous export

Bulk export of raw data to storage as soon as it arrives.

```
HTTP/1.1 200
Content-Type: application/json; charset=utf-8

{
  "Tables": [
    {
      "TableName": "Table_0",
      "Columns": [
        {
          "ColumnName": "Count",
          "DataType": "Int64",
        }
      ]
    }
  ]
}
```

# Implement Application Insights

Completed 100 XP

- 3 minutes

## Monitor

Install Application Insights in your app, set up [availability web tests](#), and:

- Set up a [dashboard](#) for your team room to keep an eye on load, responsiveness, and the performance of your dependencies, page loads, and AJAX calls.
- Discover which are the slowest and most-failing requests.
- Watch [Live Stream](#) when you deploy a new release to know immediately about any degradation.

## Detect, Diagnose

If you receive an alert or discover a problem:

- Assess how many users are affected.
- Correlate failures with exceptions, dependency calls, and traces.
- Examine profiler, snapshots, stack dumps, and trace logs.

## Build, Measure, Learn

Measure the effectiveness of each new feature that you deploy.

- Plan to measure how customers use new UX or business features.
- Write custom telemetry into your code.
- Base the next development cycle on hard evidence from your telemetry.

## Get started

Application Insights is one of the many services hosted within Microsoft Azure, and telemetry is sent there for analysis and presentation.

So, before you do anything else, you'll need a subscription to [Microsoft Azure](#).

It's free to sign up, and if you choose the basic [pricing plan](#) of Application Insights, there's no charge until your application has grown to have large usage.

If your organization already has a subscription, they could add your Microsoft account to it.

There are several ways to get started. Begin with whichever works best for you. You can add the others later.

At run time

Instrument your web app on the server. Avoids any update to the code. You need admin access to your server.

- [IIS on-premises or on a VM](#)
- [Azure web app or VM](#)

- [J2EE](#)

At development time

Add Application Insights to your code. Allows you to write custom telemetry and to instrument back-end and desktop apps.

- [Visual Studio](#) 2013 update two or later.
- [Java](#)
- [Node.js](#)
- [Other platforms](#)
- [Instrument your web pages](#) for page view, and another client-side telemetry.
- [Analyze mobile app usage](#) by integrating with Visual Studio App Center.
- [Availability tests](#) - ping your website regularly from our servers.

## Exercise - Add Application Insights to an ASP.NET core application

Completed 100 XP

- 30 minutes

Application performance management (APM) is a discipline that includes all the tools and activities involved in observing how software and hardware are doing.

These tools present performance information in the form product owners, and software development teams can use to make decisions.

Application Insights is a Microsoft Azure native APM Tool that is cross-platform. It's specialized in providing a rich & intelligent performance management toolset for Azure-hosted web apps.

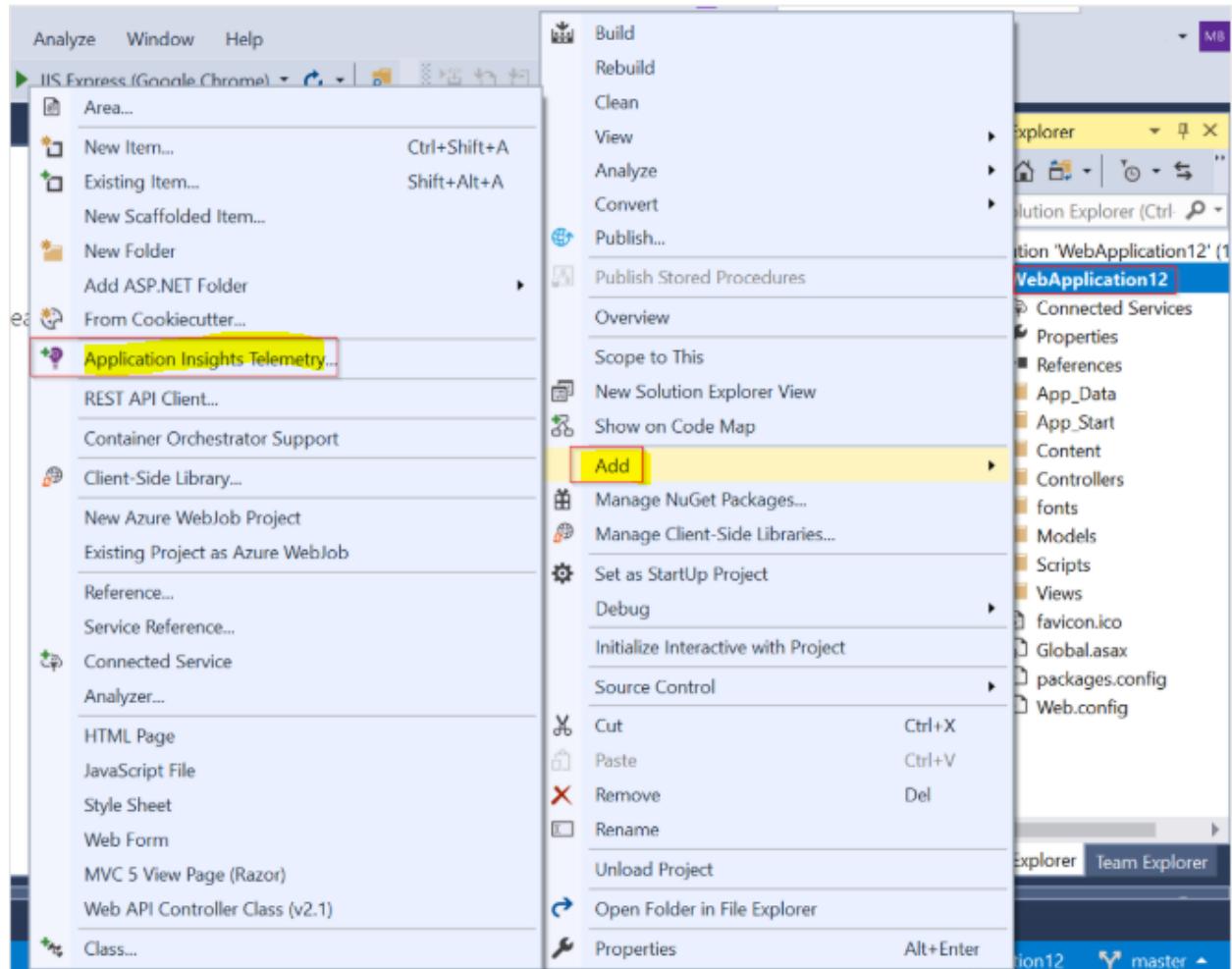
In this tutorial, we'll learn how to start with App Insights. We'll cover,

- Adding App Insights to your dotnet core app
- Accessing App Insights from within the Azure portal

## Getting started

1. To add Application Insights to your [ASP.NET](#) website, you need to:

- Install Visual Studio 2019 for Windows with the following workloads:
  - **ASP.NET** and web development (Don't uncheck the optional components)
2. In Visual Studio, create a new dotnet core project. Right-click the project and select Add, Application Insights Telemetry from the context menu.



(Depending on your Application Insights SDK version, you may be prompted to upgrade to the latest SDK release. If prompted, select Update SDK.)

3. From the Application Insights configuration screen, click Get Started to start setting up App Insights.

The screenshot shows the 'Application Insights Configuration' page for a resource named 'WebApplication12'. The title bar has tabs for 'Application Insights Configuration' and 'WebApplication12'. The main content area features a purple lightbulb icon and the heading 'Application Insights'. Below it is the tagline 'Gain insights through telemetry, analytics and smart detection'. Three sections are listed: 'Detect' (yellow lightbulb icon), 'Monitor' (blue chart and gear icon), and 'Integrate' (purple gear icon). Each section has a brief description. At the bottom is a large blue 'Get Started' button, and at the very bottom are links for 'Understand Application Insights', 'Application Insights features', and 'Privacy Statement'.

Application Insights Configuration → X WebApplication12

## Application Insights

Gain insights through telemetry, analytics and smart detection

- Detect**  
and diagnose exceptions and application performance issues
- Monitor**  
websites on Azure, hosted containers, on-premises and with other cloud providers
- Integrate**  
with your DevOps pipeline using Visual Studio, VSTS, GitHub, and web hooks

**Get Started**

[Understand Application Insights](#) | [Application Insights features](#) | [Privacy Statement](#)

4. Choose to set up a new resource group and select the location where you want the telemetry data to be persisted.

Application Insights Configuration X WebApplication12

## Application Insights

Register your app with Application Insights

Account

Microsoft

Subscription

Visual Studio Enterprise

Resource

WebApplication12 (New resource)

Configure settings...

Pricing

Visit our [pricing page](#) for details.

Register

Getting started will...

Add the AI SDK to your project

Send data to Azure

Automatically track exceptions

Automatically send publish annotations

Or just add the SDK to try local only mode

The screenshot shows the 'Application Insights Configuration' page for 'WebApplication12'. It includes fields for Account (Microsoft), Subscription (Visual Studio Enterprise), and Resource (WebApplication12). A red box highlights the 'Configure settings...' button. Below, a 'Register' button is also highlighted with a red box. The page provides links for getting started with the AI SDK and for trying local mode. It also lists four automatic tracking options: adding the AI SDK to the project, sending data to Azure, automatically tracking exceptions, and automatically sending annotations.

## Summary

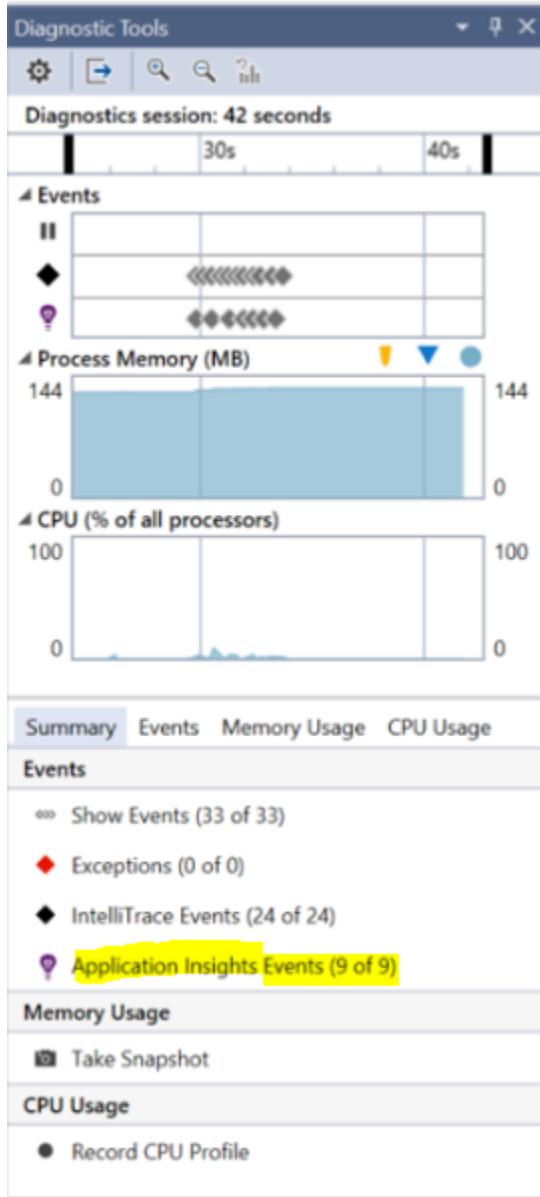
So far, we have added App Insights in a dotnet core application.

The Application Insights getting started experience allows you to create a new resource group in the wished location where the App Insights instance gets created.

The instrumentation key for the app insights instance is injected into the application configuration automatically.

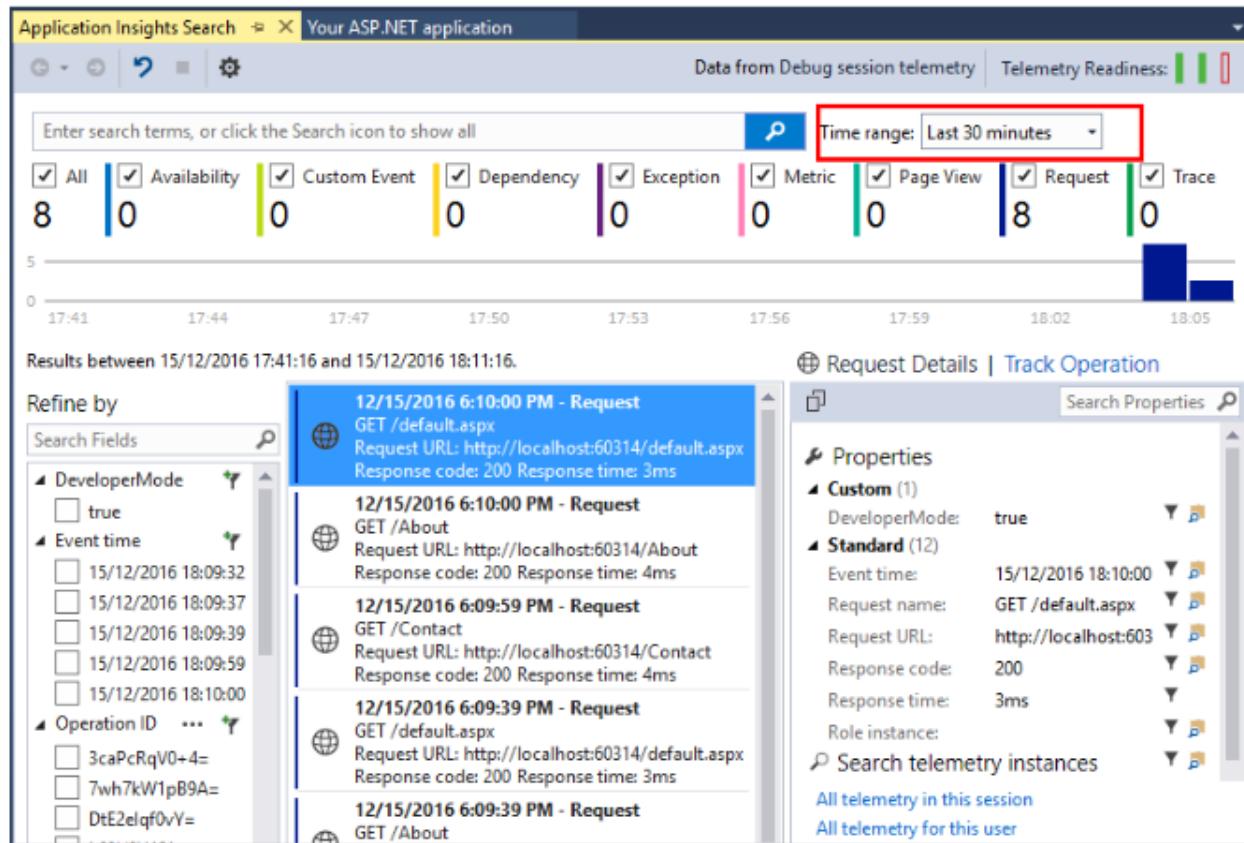
## How to do it

1. Run your app with F5. Open different pages to generate some telemetry. In Visual Studio, you'll see a count of the events that have been logged.



2. You can see your telemetry in Visual Studio or the Application Insights web portal. Search telemetry in Visual Studio to help you debug your app. Monitor performance and usage in the web portal when your system is live. In Visual Studio, view Application Insights data. Select Solution Explorer > Connected Services > Right-click Application Insights, then click Search-Live Telemetry.

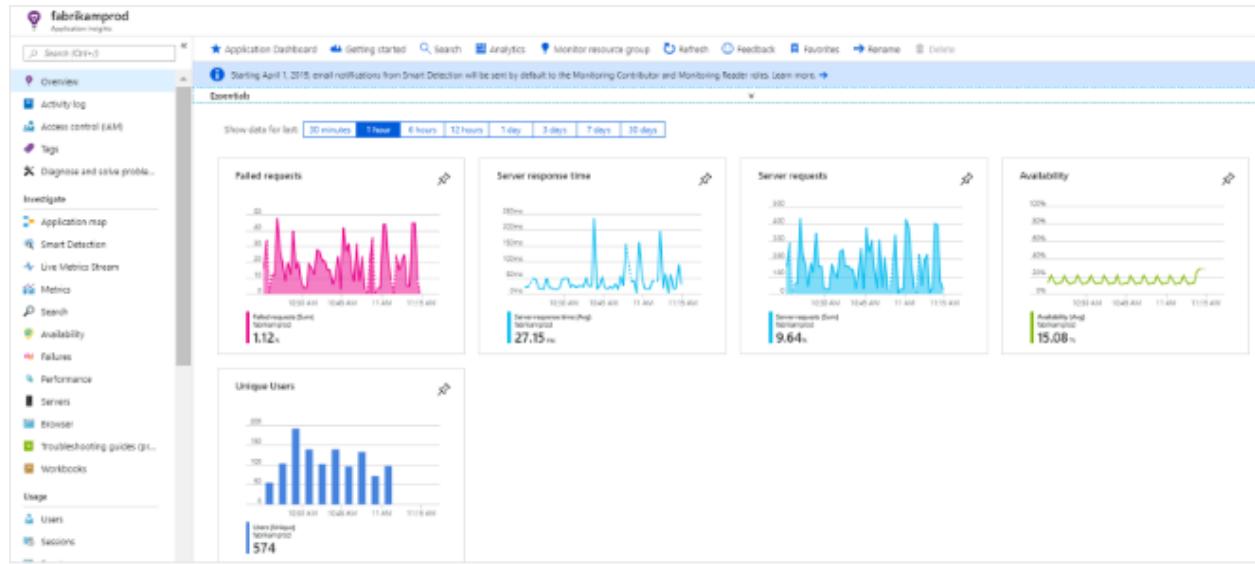
In the Visual Studio Application Insights Search window, you'll see the data from your application for telemetry generated on the server side of your app. Experiment with the filters, and click any event to see more detail.



3. You can also see telemetry in the Application Insights web portal (unless you choose to install only the SDK). The portal has more charts, analytic tools, and cross-component views than Visual Studio. The portal also provides alerts.

Open your Application Insights resource. Either sign into the Azure portal and find it there, or select Solution Explorer > Connected Services > right-click Application Insights > Open Application Insights Portal and let it take you there.

The portal opens on a view of the telemetry from your app.



## How it works

Application Insights configures your application's unique key (AppInsights Key). The Application Insights SDK uses this key to identify the Azure App Insights workspace the telemetry data needs to be uploaded. The SDK and the key are merely used to pump the telemetry data points out of your application. The heavy lifting of data correlation, analysis, and insights is done within Azure.

## There's more

This tutorial taught us how to add Application Insights to your dotnet core application.

App Insights offers a wide range of features.

You can learn more about these at [Start Monitoring Your ASP.NET Core Web Application](#).

# Monitor application performance with Application Insights

Completed 100 XP

- 60 minutes

**Estimated time:** 60 minutes.

**Lab files:** none.

## Scenario

Application Insights is an extensible Application Performance Management (APM) service for web developers on multiple platforms. You can use it to monitor your live web applications. It automatically detects performance anomalies, includes powerful analytics tools to help you diagnose issues, and helps you continuously improve performance and usability. It works for apps on various platforms, including .NET, Node.js, and Java EE, hosted on-premises, hybrid, or any public cloud. It integrates with your DevOps process with connection points available in various development tools. It also allows you to monitor and analyze telemetry from mobile apps through integration with Visual Studio App Center.

In this lab, you'll learn how to add Application Insights to an existing web application and monitor the application via the Azure portal.

## Objectives

After completing this lab, you'll be able to:

- Deploy Azure App Service web apps.
- Generate and monitor Azure web app application traffic by using Application Insights.
- Investigate Azure web app performance by using Application Insights.
- Track Azure web app usage by using Application Insights.
- Create Azure web app alerts by using Application Insights.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).
- Identify an existing Azure subscription or create a new one.
- Verify that you have a Microsoft or Microsoft Entra account with the **Owner** role in the Azure subscription and the **Global Administrator** role in the Microsoft Entra

tenant associated with the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](#) and [View and assign administrator roles in Microsoft Entra ID](#).

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Monitor an Azure App Service web app using Azure Application Insights.
- Exercise 2: Remove the Azure lab resources.

# Manage alerts, blameless retrospectives and a just culture

## Examine when get a notification

Completed 100 XP

- 2 minutes

[Application Insights](#) automatically analyzes the performance of your web application and can warn you about potential problems. You might be reading it because you received one of our smart detection notifications.

This feature requires no particular setup other than configuring your app for Application Insights (on [ASP.NET](#), [Java](#), or [Node.js](#), and [web page code](#)). It's active when your app generates enough telemetry.

### When would I get a smart detection notification?

Application Insights has detected that the performance of your application has degraded in one of these ways:

- **Response time degradation** - Your app has started responding to requests more slowly than it used to. The change might have been rapid, for example, because there was a regression in your latest deployment. Or it might have been gradual, maybe caused by a memory leak.
- **Dependency duration degradation** - Your app makes calls to a REST API, database, or other dependencies. The dependency is responding more slowly than it used to.
- **Slow performance pattern** - Your app appears to have a performance issue that is affecting only some requests. For example, pages are loading more slowly on one type of browser than others; or requests are being served more slowly from one server. Currently, our algorithms look at page load times, request response times, and dependency response times.

Smart Detection requires at least eight days of telemetry at a workable volume to establish a normal performance baseline. So, after your application has been running for that period, any significant issue will result in a notification.

### Does my app have a problem?

No, a notification doesn't mean that your app has a problem. It's simply a suggestion about something you might want to look at more closely.

## Explore how to fix it

Completed 100 XP

- 1 minute

The notifications include diagnostic information. Here's an example:

## Server response time degradation

fabrikamprod

Send a smile Send a frown New Work Item View Work Items More

Was this detection helpful? Please send us a smile or a frown

### Detection Properties

|                        |                                     |
|------------------------|-------------------------------------|
| Rule name              | Degradation in server response time |
| When                   | 3/23 2:00 AM - 3/24 1:59 AM         |
| Operation name         | GET Home/Index                      |
| Detected response time | 1.62 sec                            |
| Normal response time   | 0.524 sec                           |

### Detection Analysis

|                |    |
|----------------|----|
| Affected users | 76 |
|----------------|----|

#### Server response time

| Date   | Avg Response Time (s) | 90th Percentile (s) |
|--------|-----------------------|---------------------|
| Mar 16 | 0.4                   | 1.5                 |
| Mar 17 | 0.3                   | 1.0                 |
| Mar 18 | 0.4                   | 1.6                 |
| Mar 19 | 0.3                   | 1.0                 |
| Mar 20 | 0.4                   | 1.1                 |
| Mar 21 | 0.3                   | 1.6                 |
| Mar 22 | 1.4                   | 2.8                 |

#### Server requests

| Date   | Server Requests (K) |
|--------|---------------------|
| Mar 16 | 2.2                 |
| Mar 17 | 2.2                 |
| Mar 18 | 2.2                 |
| Mar 19 | 2.2                 |
| Mar 20 | 2.2                 |
| Mar 21 | 2.2                 |
| Mar 22 | 2.2                 |

#### Degradation in related dependency duration

| Date   | Degradation Duration (ms) |
|--------|---------------------------|
| Mar 16 | 0                         |
| Mar 17 | 0                         |
| Mar 18 | 0                         |
| Mar 19 | 0                         |
| Mar 20 | 0                         |
| Mar 21 | 0                         |
| Mar 22 | 220                       |

### Related items and reports

- Diagnose example profiler traces
- Diagnose response times (8 days)
- View requests with this Operation name (24 hours)
- View failed requests with this Operation name

- Triage. The notification shows you how many users or how many operations are affected. It can help you assign a priority to the problem.
- Scope. Is the problem affecting all traffic, or just some pages? Is it restricted to particular browsers or locations? This information can be obtained from the notification.
- Diagnose. Often, the diagnostic information in the notification will suggest the nature of the problem. For example, if response time slows down when the request rate is high, that means your server or dependencies are overloaded. Otherwise, open the Performance blade in Application Insights. There, you'll find [Profiler](#) data. If exceptions are thrown, you can also try the [snapshot debugger](#).

## Explore smart detection notifications

Completed 100 XP

- 1 minute

Smart detection notifications are enabled by default and sent to [owners, contributors, and readers access to the Application Insights resource](#).

To change it, click Configure in the email notification, or open Smart Detection settings in Application Insights.

| NAME                                | SEVERITY    |
|-------------------------------------|-------------|
| Slow page load time                 | Information |
| Slow server response time           | Information |
| Long dependency duration            | Information |
| Degradation in server response time | Information |
| Azure cloud service issues          | Information |
| Degradation in dependency duration  | Information |
| Failure Anomalies - fabrikamprod    | Alert       |

- You can use the unsubscribe link in the smart detection email to stop receiving the email notifications.

Emails about smart detections performance anomalies are limited to one email per day per Application Insights resource.

The email would be sent only if at least one new issue was detected on that day. You won't get repeats of any message.

# Improve performance

Completed 100 XP

- 3 minutes

Slow and failed responses are one of the biggest frustrations for website users, as you know from your own experience. So, it's essential to address the issues.

## Triage

- First, does it matter? If a page is always slow to load, but only 1% of your site's users ever have to look at it, maybe you have more important things to think about. On the other hand, if only 1% of users open it, but it throws exceptions every time, that might be worth investigating. Use the impact statement (affected users or % of traffic) as a general guide but be aware that it isn't the whole story. Gather other evidence to confirm. Consider the parameters of the issue. If it's geography-dependent, set up [availability tests](#) including that region: there might be network issues in that area.

## Diagnose slow page loads

- Where is the problem? Is the server slow to respond, is the page long, or does the browser have to do much work to display it? Open the Browsers metric blade. The segmented display of browser page load time shows where the time is going.
  - If Send Request Time is high, the server responds slowly, or the request is a post with much data. Look at the [performance metrics](#) to investigate response times.
  - Set up [dependency tracking](#) to see whether the slowness is because of external services or your database.
  - If Receiving Response is predominant, your page and its dependent parts - JavaScript, CSS, images, and so on (but not asynchronously loaded data) are long. Set up an [availability test](#) and be sure to set the

- option to load dependent parts. When you get some results, open the detail of a result, and expand it to see the load times of different files.
- High Client Processing time suggests scripts are running slowly. If the reason isn't clear, consider adding some timing code and sending the times in track metrics calls.

## Improve slow pages

- There's a web full of advice on improving your server responses and page load times, so we won't try to repeat it all here. Here are a few tips that you probably already know about to get you thinking:
  - Slow loading because of large files: Load the scripts and other parts asynchronously. Use script bundling. Break the main page into widgets that load their data separately. Don't send plain old HTML for long tables: use a script to request the data as JSON or another compact format, then fill the table in place. There are remarkable frameworks to help with all of it. (They also involve great scripts)
  - Slow server dependencies: Consider the geographical locations of your components. For example, if you use Azure, ensure the web server and the database are in the same region. Do queries retrieve more information than they need? Would caching or batching help?
  - Capacity issues: Look at the server metrics of response times and request counts. If response times peak disproportionately with peaks in request counts, your servers are likely stretched.

## Understand server response time degradation

Completed 100 XP

- 1 minute

The response time degradation notification tells you:

- The response time compared to normal response time for this operation.
- How many users are affected?
- Average response time and 90th percentile response time for this operation on the day of the detection and seven days before.
- Count of this operation requests on the day of the detection and seven days before.

- Correlation between degradation in this operation and degradations in related dependencies.
  - Links to help you diagnose the problem.
    - Profiler traces to help you view where operation time is spent (the link is available if Profiler trace examples were collected for this operation during the detection period).
    - Performance reports in Metric Explorer, where you can slice and dice time range/filters for this operation.
    - Search for this call to view specific call properties.
    - Failure reports - If count > 1 it means that there were failures in this operation that might have contributed to performance degradation.

## Reduce meaningless and non-actionable alerts

Completed 100 XP

- 2 minutes

Monitoring and alerting enable a system to tell us when it's' broken, or perhaps to tell us what is about to break.

**When the system can't automatically fix itself, we want a human to investigate the alert.**  
 Determine if there's a real problem at hand, mitigate the problem, and determine the root cause.

Unless you're doing security auditing on narrowly scoped components of a system, you should never trigger an alert simply because "something seems a bit weird."

When you're reviewing existing alerts or writing new-alerting rules, consider these things to keep your alerts relevant and your on-call rotation happier:

- Alerts that trigger call-out should be urgent, important, actionable, and **real**.
- They should represent either ongoing or imminent problems with your service.
- Err on the side of removing noisy alerts – over-monitoring is a more-challenging problem to solve under-monitoring.
- You should almost always classify the problem into availability & basic functionality; latency; correctness (completeness, freshness, and durability of data); and feature-specific problems.

- Symptoms are a better way to capture more problems more comprehensively and robustly with less effort.
- Include cause-based information in symptom-based pages or on dashboards but avoid alerting directly on causes.
- The further up your serving stack you go, the more distinct problems you catch in a single rule. But don't go so far you can't sufficiently distinguish what is going on.
- If you want an on-call rotation, it's imperative to have a system for dealing with things that need a timely response but aren't imminently critical.

## Examine blameless retrospective

Completed 100 XP

- 4 minutes

### What does it mean to have a blameless retrospective?

Anyone who has worked with technology at any scale is familiar with failure.

Failure cares not about the architecture designs you drudge over, the code you write and review, or the alerts and metrics you meticulously pore through. So, failure happens.

It's a foregone conclusion when working with complex systems. But what about those failures that have resulted from individuals' actions (or lack of action, in some cases)? What do you do with those careless humans who caused everyone a bad day?

Maybe they should be fired. Or perhaps they need to be prevented from touching the dangerous bits again. Or they need more training. It's the traditional view of "human error," which focuses on the individual's characteristics. Also called the "Bad Apple Theory" – get rid of the bad apples and the human error.

It seems simple, right? Organizations that have pioneered DevOps are shying away from this traditional view. Instead, these DevOps-practicing organizations want to view mistakes, slips, lapses, etc., from a learning perspective. Having a blameless Post-mortem on outages and accidents is part of it.

What does it mean to have a 'blameless' retrospective? Does it mean everyone gets off the hook for making mistakes? No.

Well, maybe. It depends on what "gets off the hook" means. Let me explain.

Having a **Just Culture** means that you're making an effort to balance safety **and** accountability. It means that by investigating mistakes, focusing on the situational aspects of a failure's mechanism.

In the decision-making process of individuals proximate to the failure, an organization can come out safer than it would usually be if it had punished the actors involved as remediation.

Having a "blameless" retrospective process means that engineers whose actions have contributed to an accident can give a detailed account of:

- What actions do they take at what time?
- What effects do they observe?
- Expectations they had.
- Assumptions they had made.
- Their understanding of the timeline of events as they occurred.

And that they can give this detailed account **without fear of punishment or retribution.**

Why shouldn't engineers be punished or reprimanded? Because if they fear being blamed, they might not provide the necessary details to understand the failure's mechanism, pathology, and operation.

This lack of understanding of the accident's occurrence guarantees that it **will** be repeated. If not with the original engineer, another one in the future.

If we use "blame" as the predominant approach, we implicitly accept that *deterrence* is how organizations become safer.

This is founded on the belief that individuals, not situations, cause errors.

It's also aligned with the idea there must be some fear that **not doing** one's job correctly could lead to punishment because the fear of punishment will motivate people to act correctly in the future. Right?

This cycle of name/blame/shame can be looked at like this:

- Engineer acts and contributes to a failure or incident.
- Engineer is punished, shamed, blamed, or retrained.

- Reduced trust between engineers on the ground (the "sharp end") and management (the "blunt end") looking for someone to scapegoat.
- Engineers become silent on details about actions/situations/observations, resulting in "Cover-Your-Mistake" engineering (from fear of punishment)
- Management becomes less aware and informed on how work is being performed daily. Engineers become less educated on lurking or latent conditions for failure because of the silence mentioned in the previous four.
- Errors are more likely, and latent conditions can't be identified because of the previous five.
- Repeat the first step.

We need to avoid this cycle. We want the engineer who has made an error to explain why (either explicitly or implicitly) they did what they did and why the action made sense to them then.

It's paramount to understand the pathology of the failure. The action made sense to the person when they took it because if it had not made sense, they **wouldn't have taken action in the first place.**

The base fundamental here's something [Erik Hollnagel](#) has said:

*We must strive to understand that accidents don't happen because people gamble and lose.*

*Accidents happen because the person believes that:*

*...what is about to happen isn't possible,  
...or what is about to happen has no connection to what they're doing,  
...or that the possibility of getting the intended outcome is worth whatever risk there is.*

## Develop a just culture

Completed 100 XP

- 2 minutes

A funny thing happens when engineers make mistakes and feel safe when giving details about it: they aren't only willing to be held accountable, but they're also enthusiastic about helping the rest of the company avoid the same error in the future.

They are, after all, the most expert in their error.

They ought to be heavily involved in coming up with remediation items.

So technically, engineers aren't at all "off the hook" with a blameless PostMortem process. They're very much on the hook for helping become safer and more resilient in the end. And lo and behold: most engineers I know find this idea of making things better for others a worthwhile exercise.

So, what do we do to enable a "Just Culture"?

- Encourage learning by having these blameless Postmortems on outages and accidents.
- The goal is to understand \*\*how \*\*an accident could have happened, to better equip ourselves from it happening in the future.
- Gather details from multiple perspectives on failures, and don't punish people for making mistakes.
- Instead of punishing engineers, we give them the requisite authority to improve safety by providing detailed accounts of their contributions to failures.
- Enable and encourage people who *make mistakes* to *educate* the rest of the organization on how not to make them in the future.
- Accept that there's always a discretionary space where humans can decide to make actions or not and that the judgment of those decisions lies in hindsight.
- Accept that the **Hindsight Bias** will continue to cloud our assessment of past events and work hard to eliminate it.
- Accept that the **Fundamental Attribution Error** is also difficult to escape, so we focus on the environment and circumstances people are working in when investigating accidents.
- Strive to make sure that the blunt end of the organization understands how work is getting done (as opposed to how they imagine it's getting done via Gantt charts and procedures) on the sharp end.
- The sharp end is relied upon to inform the organization of the line between appropriate and inappropriate behavior. It isn't something that the blunt end can come up with on its own.

Failure happens. To understand how failures occur, we first must understand our **reactions** to failure.

One option is to assume the single cause is incompetence and scream at engineers to make them "pay attention!" or "be more careful!"

Another option is to take a hard look at how the accident happened, treat the engineers involved with respect, and *learn* from the event.

For more information, see also:

- [Brian Harry's Blog - A good incident postmortem](#)

# Design processes to automate application analytics

## Explore rapid responses and augmented search

Completed 100 XP

- 4 minutes

In an Agile environment, you may typically find multiple development teams that work simultaneously. Introducing new code or code changes daily and sometimes several times a day.

In such an immediate environment, it's prevalent to find problems that have "slipped through the cracks" to find themselves in the production environment. When these issues arise, they have probably already-impacted end users, requiring a speedy resolution. It means that teams must conduct a rapid investigation to identify the root cause of the problem.

Identifying where these symptoms are coming from and then isolating the root cause is a challenging task. Symptoms can be found across various layers of a large hybrid IT environment, such as different servers/VMs, storage devices, databases, to the front-end and server-side code. Investigations that traditionally would take hours or days to complete must be completed within minutes.

Teams must examine the infrastructure and application logs as part of this investigation process. However, the massive amount of log records produced in these environments makes it impossible to do this manually.

It's much like trying to find a needle in a haystack. In most cases, these investigations are conducted using log management and analysis systems that collect and aggregate these logs (from infrastructure elements and applications), centralizing them in a single place and then-providing search capabilities to explore the data.

These solutions make it possible to conduct the investigation, but they still rely entirely on investigation skills and user knowledge. The user must know exactly what to search for and have a deep understanding of the environment to use them effectively.

It's crucial to understand that the log files of applications are far less predictable than the log files of infrastructure elements. The errors are essentially messages and error numbers that have been introduced to the code by developers in a non-consistent manner.

So, search queries yield thousands of results in most cases and do not include important ones, even when the user is skilled. That leaves the user with the same "needle in the haystack" situation.

## Assisting DevOps with augmented Search

A new breed of log management and analysis technologies has evolved to solve this challenge. These technologies facilitate the identification and investigation processes using Augmented Search.

Explicitly designed to deal with application logs' chaotic and unpredictable nature, Augmented Search considers that users don't necessarily know what to search for, especially in the chaotic application layer environment.

The analysis algorithm automatically identifies errors, risk factors, and problem indicators while analyzing their severity by combining semantic processing, statistical models, and machine learning to analyze and "understand" the events in the logs. These insights are displayed as intelligence layers on top of the search results, helping the user quickly discover the most relevant and essential information.

Although DevOps engineers may be familiar with the infrastructure and system architecture, the data is constantly changing with continuous fast-paced deployment cycles and constant code changes. It means that DevOps teams can use their intuition and knowledge to start investigating each problem, but they have blind spots that consume time because of the dynamic nature of the log data.

Combining the decisions that DevOps engineers make during their investigation with the Augmented Search engine information layers on the critical problems that occurred during the period of interest can help guide them through these blind spots quickly.

Combining the user's intellect, acquaintance with the system's architecture, and Augmented Search machine-learning capabilities on the dynamic data makes it faster and easier to focus on the most relevant data. Here's how that works in practice: One of the servers went down, and any attempt to reinitiate the server has failed. However,

since the process is running, the server seems to be up. In this case, end users are complaining that an application isn't responding.

This symptom could be related to many problems in a complex environment with many servers. Focusing on the server behind this problem can be difficult, as it seems to be up. But finding the root cause of the problem requires a lengthy investigation, even when you know which server is behind this problem.

Augmented Search will display a layer that highlights critical events during the specified period instead of going over thousands of search results. These highlights provide information about the sources of the events, assisting in the triage process.

At this point, DevOps engineers can understand the impact of the problem (for example, which servers are affected by it) and then continue the investigation to find the root cause of these problems. Using Augmented Search, DevOps engineers can identify a problem and the root cause in a matter of seconds instead of examining thousands of log events or running multiple checks on the various servers.

Adding this type of visibility to log analysis and the ability to surface critical events out of tens of thousands - and often millions - of events is essential in a fast-paced environment that constantly introduces changes.

## Integrate telemetry

Completed 100 XP

- 4 minutes

A key factor to automating feedback is telemetry. By inserting telemetric data into your production application and environment, the DevOps team can automate feedback mechanisms while monitoring applications in real time.

DevOps teams use telemetry to see and solve problems as they occur, but this data can be helpful to both technical and business users.

When properly instrumented, telemetry can also be used to see and understand how customers are engaging with the application in real time.

It could be critical information for product managers, marketing teams, and customer support. So, feedback mechanisms must share continuous intelligence with all stakeholders.

## What is telemetry, and why should I care?

In the software development world, telemetry can offer insights on which features end users use most, detect bugs and issues, and provide better visibility into the performance without asking for feedback directly from users.

In DevOps and the world of modern cloud apps, we're tracking the health and performance of an application.

That telemetry data comes from application logs, infrastructure logs, metrics, and events.

The measurements are things like memory consumption, CPU performance, and database response time.

Events can be used to measure everything else, such as when a user logged in, when an item is added to a basket, when a sale is made, and so on.

The concept of telemetry is often confused with just logging. But logging is a tool used in the development process to diagnose errors and code flows. It's focused on the internal structure of a website, app, or another development project. Logging only gives you a single dimension view. With insights into infrastructure logs, metrics, and events, you have a 360-degree view of understanding user intent and behavior.

Once a project is released, telemetry is what you are looking for to enable data collection from real-world use.

Telemetry is what makes it possible to collect all that raw data that becomes valuable, actionable analytics.

## Benefits of telemetry

The primary benefit of telemetry is the ability of an end user to monitor the state of an object or environment while physically far removed from it.

Once you've shipped a product, you can't be physically present, peering over the shoulders of thousands (or millions) of users as they engage with your product to find out what works, what is easy, and what is cumbersome.

Thanks to telemetry, those insights can be delivered directly into a dashboard for you to analyze and act.

Because telemetry provides insights into how well your product is working for your end users – as they use it – it's a unique tool for ongoing performance monitoring and management.

Plus, you can use the data you've gathered from version 1.0-to-drive improvements and prioritize updates for your release of version 2.0.

Telemetry enables you to answer questions such as:

- Are your customers using the features you expect? How are they engaging with your product?
- How frequently are users engaging with your app, and for what duration?
- What settings options do users select most? Do they prefer certain display types, input modalities, screen orientation, or other device configurations?
- What happens when crashes occur? Are crashes happening more frequently when certain features or functions are used? What is the context surrounding a crash?

The answers to these and the many other questions that can be answered with telemetry are invaluable to the development process.

It will enable you to make continuous improvements and introduce new features that, to your end users, may seem as though you have been reading their minds – which you've been, thanks to telemetry.

## Challenges of telemetry

Telemetry is a fantastic technology, but it isn't without its challenges.

The most prominent challenge – and a commonly occurring issue – isn't with telemetry itself but with your end users and their willingness to allow what some see as Big Brother-Esque spying.

In short, some users immediately turn it off when they notice it, meaning any data generated from their use of your product won't be gathered or reported.

That means the experience of those users won't be accounted for when it comes to planning your future roadmap, fixing bugs, or addressing other issues in your app.

Although it isn't necessarily a problem by itself, the issue is that users who tend to disallow these types of technologies can tend to fall into the more tech-savvy portion of your user base.

It can result in the dumbing-down of software. On the other hand, other users take no notice of telemetry happening behind the scenes or ignore it if they do.

It's a problem without a clear solution—and it doesn't negate the overall power of telemetry for driving development—but one to keep in mind as you analyze your data.

So, when designing a strategy for how you consider the feedback from application telemetry, it's necessary to account for users who don't participate in providing the telemetry.

## Examine monitoring tools and technologies

Completed 100 XP

- 2 minutes

Continuous monitoring of applications in production environments is typically implemented with application performance management (APM) solutions that intelligently monitor, analyze, and manage cloud, on-premises, and hybrid applications and IT infrastructure.

These APM solutions enable you to monitor your users' experience and improve the stability of your application infrastructure. It helps identify the root cause of issues quickly to prevent outages and keep users satisfied proactively.

With a DevOps approach, we also see more customers broaden the scope of continuous monitoring into the staging, testing, and even development environments. It's possible because development and test teams following a DevOps approach are striving to use production-like environments for testing as much as possible.

By running APM solutions earlier in the life cycle, development teams get feedback about how applications will eventually do in the production and take corrective action

**much earlier.** Also, operations teams advising the development teams get advanced knowledge and experience to better prepare and tune the production environment, resulting in far more stable releases into production.

Applications are more business-critical than ever. They must always be up, always fast, and constantly improving. Embracing a DevOps approach will allow you to reduce your cycle times to hours instead of months, but you must keep ensuring a great user experience!

Continuous monitoring of your entire DevOps life cycle will ensure development and operations teams collaborate to optimize the user experience every step of the way, leaving more time for your next significant innovation.

When shortlisting a monitoring tool, you should seek the following advanced features:

- **Synthetic Monitoring:** Developers, testers, and operations staff all need to ensure that their internet and intranet-mobile applications and web applications are tested and operate successfully from different points of presence worldwide.
- **Alert Management:** Developers, testers, and operations staff all need to send notifications via email, voice mail, text, mobile push notifications, and Slack messages when specific situations or events occur in development, testing, or production environments, to get the right people's attention and to manage their response.
- **Deployment Automation:** Developers, testers, and operations staff use different tools to schedule and deploy complex applications and configure them in development, testing, and production environments. We'll discuss the best practices for these teams to collaborate effectively and efficiently and avoid potential duplication and erroneous information.
- **Analytics:** Developers need to look for patterns in log messages to identify if there's a problem in the code. Operations need to do root cause analysis across multiple log files to identify the source of the problem in complex applications and systems.

# Share knowledge within teams

## Share acquired knowledge within development teams

Completed 100 XP

- 1 minute

Organizational knowledge builds up within development teams over time.

It's essential to avoid endlessly relearning the same lessons that the team (and the organization) has learned before.

As staff turnover occurs in teams, it's easy for organizational knowledge to be lost. It's important to record this knowledge to avoid this loss.

A simple example is if the team has developed coding standards. It should be captured and not just exist as word of mouth.

Relearning old lessons is both wasteful and expensive.

## To reflect

Azure DevOps can be used with a wide range of existing tooling used to share knowledge.

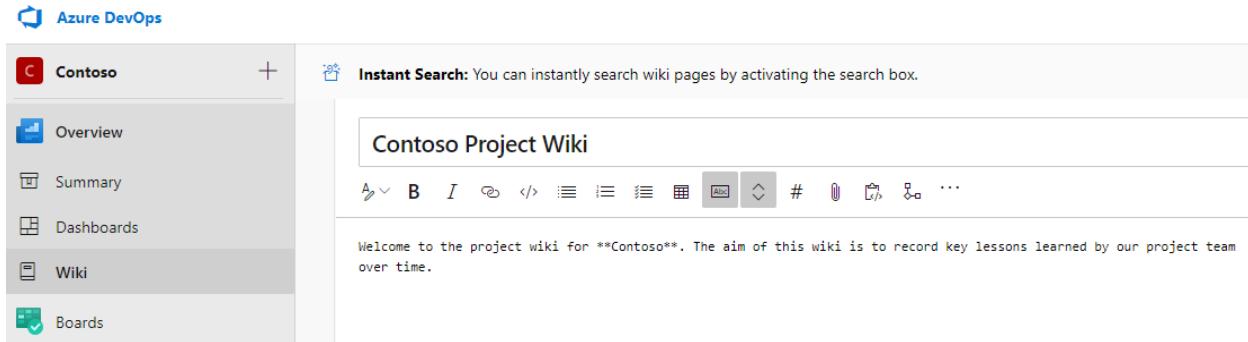
- Which knowledge-sharing tools do you currently use (if any)?
- What do you or don't you like about the tools?

# Introduction to Azure DevOps project wikis

Completed 100 XP

- 2 minutes

Azure DevOps Projects includes an option to create a project wiki.



The wiki to share information with your team to understand and contribute to your project.

Wikis are stored in a repository. No wiki is automatically provisioned.

## Prerequisites

You must have permission to **Create a Repository** to publish code as a wiki. While the **Project Administrators** group has this permission by default, it can be assigned to others.

To add or edit wiki pages, you should be a member of the **Contributors** group.

All team project members (including stakeholders) can view the wiki.

## Creation

The following article includes details on creating a wiki: [Create a Wiki for your project](#).

## Editing the wiki

The following article includes details on publishing a Git repository to a wiki: [Publish a Git repository to a wiki](#).

## Markdown

Azure DevOps Wikis are written in Markdown and can also include file attachments and videos.

Markdown is a markup language. The plain text includes formatting syntax. It has become the de facto standard for writing projects and software documentation.

One key reason for this is that because it's made up of plain text, it's easier to merge in the same way that program code is merged.

It allows documents to be managed with the same tools used to create other code in a project.

## GitHub Flavored Markdown (GFM)

GFM is a formal specification released by GitHub that added extensions to a base format called CommonMark. GFM is widely used both within GitHub and externally. GFM is rendered in Azure DevOps Wikis.

## Mermaid

Mermaid has become an essential extension to Markdown because it allows diagrams to be included in the documentation.

It overcomes the difficulties of merging documentation, including diagrams represented as binary files.

Welcome to the project wiki for \*\*Contoso\*\*. The aim of this wiki is to record key lessons learned by our project team over time.

```
::: mermaid
graph LR;
A[Wiki supports Mermaid] --> B[Visit https://mermaidjs.github.io/ for Mermaid syntax];
:::
```

Welcome to the project wiki for **Contoso**. The aim of this wiki is to record key lessons learned by our project team over time.

Wiki supports Mermaid

→ Visit <https://mermaidjs.github.io/> for Mermaid syntax

Details on Mermaid syntax can be found here: [Mermaid Introduction](#).

# Integrate with Azure Boards

Completed 100 XP

- 7 minutes

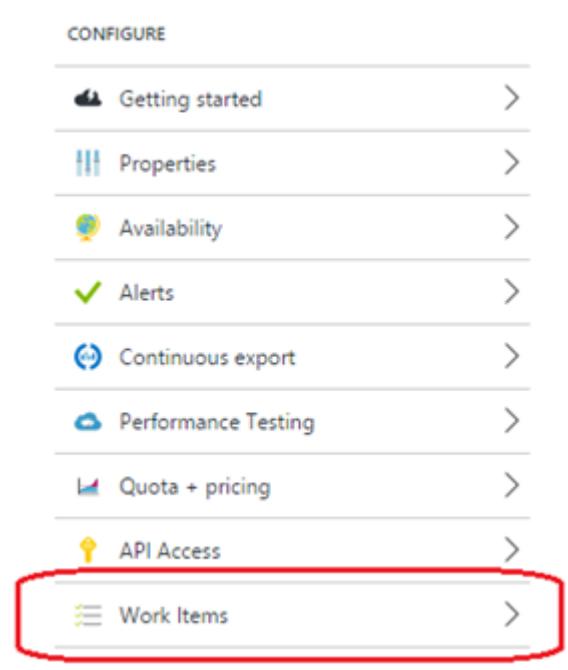
Work item integration functionality allows you to easily create work items in Azure DevOps with relevant Application Insights data embedded in them.

Configuring this association and creating work items is straightforward (this process should only take a minute or two).

## Configuring work item integration

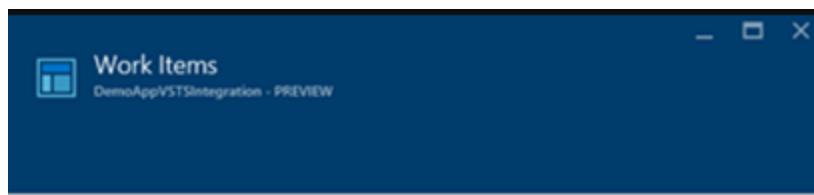
To configure work item integration for an Application Insights resource, navigate to your settings blade for that resource.

You'll note that a new item in the "Configure" section of the settings blade says, "Work Items."



Click on it, and the configuration blade for work items will open.

All you need to do is fill out the information about the Azure DevOps system to which you want to connect, along with the project where you want to write your work items:



Set up tracking system configuration to create work items from Search events or Proactive detection events.

Tracking System

Visual Studio Team Services

\* URL ?

\* Project Collection ?

\* Project ?

Authorization

\* Required ?

Advanced Configuration ?

\* Area ?

Work Item Type ?

Assigned To ?

By selecting OK, you are authorizing this resource to create or edit work items with the configured tracking system.

Once that information is in place, you can click on the Authorization button, where you'll be redirected to authorize access in your selected Azure DevOps system so that work items can be written there:

## Authorize application

 **AppInsights by AppInsights**  
Authorize AppInsights to read and write work items on your behalf.  
[Visit application's website](#)

Review requested permissions

**Work items (read and write)**  
Grants the ability to read, create, and update work items and queries, update board metadata, read area and iterations paths other work item tracking related metadata, execute queries, and to receive notifications about work item events via service hooks.

[Learn more](#)

If you change your mind at any time, you can manage authorizations on your [profile page](#).

**Accept**   **Deny**

By clicking **Accept**, you agree to AppInsights [Terms of Use](#) and [Privacy Statement](#).

Once you've completed the authorization process, you can set defaults for "area path" and "assigned to."

Only an area path is required (if you have not set up specific area paths in your project, that is ok. Use the project's name, as it's the top-level area path.

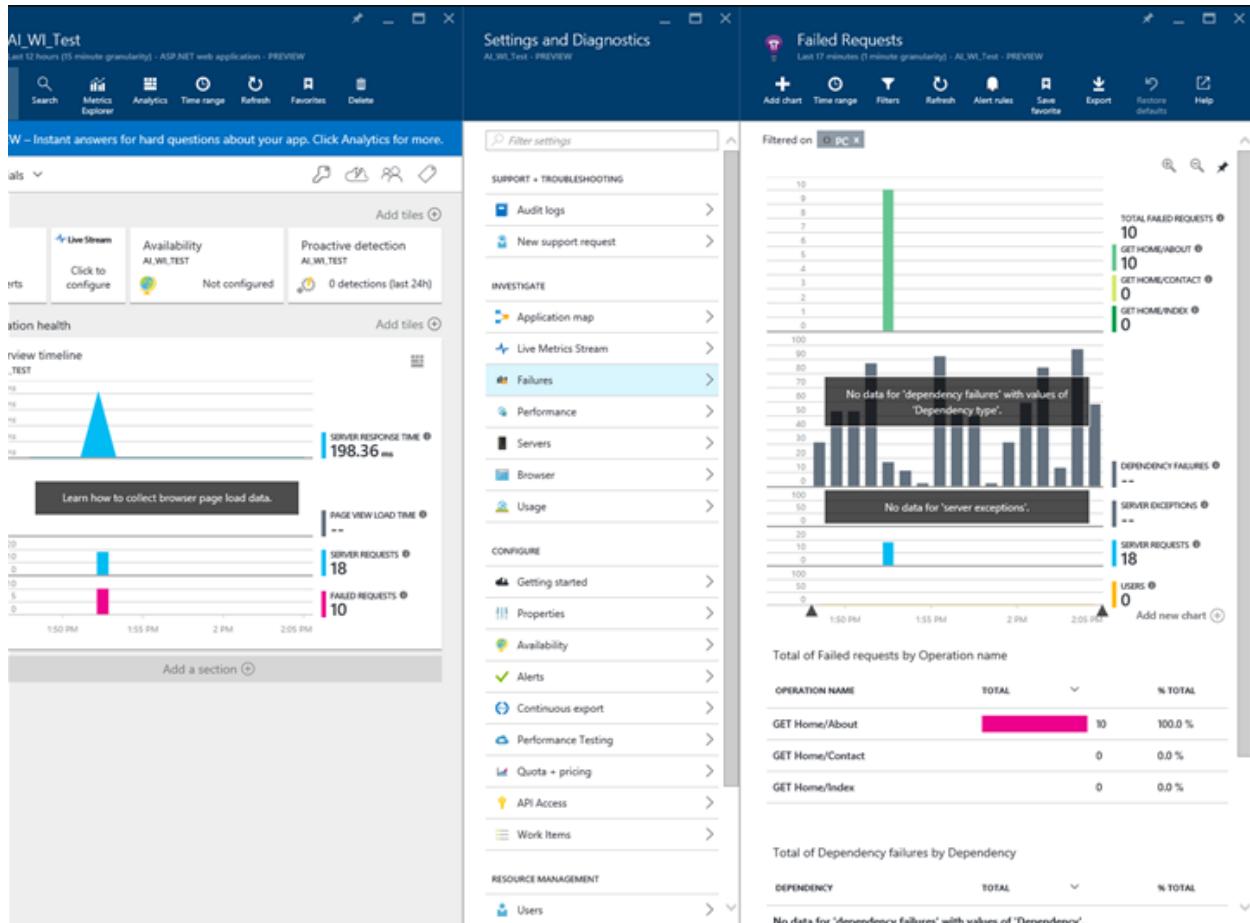
Click OK, and assuming you have entered everything correctly; you'll see a message stating "Validation Successful," and the blade will close. You're now ready to start creating work items!

## Creating work items

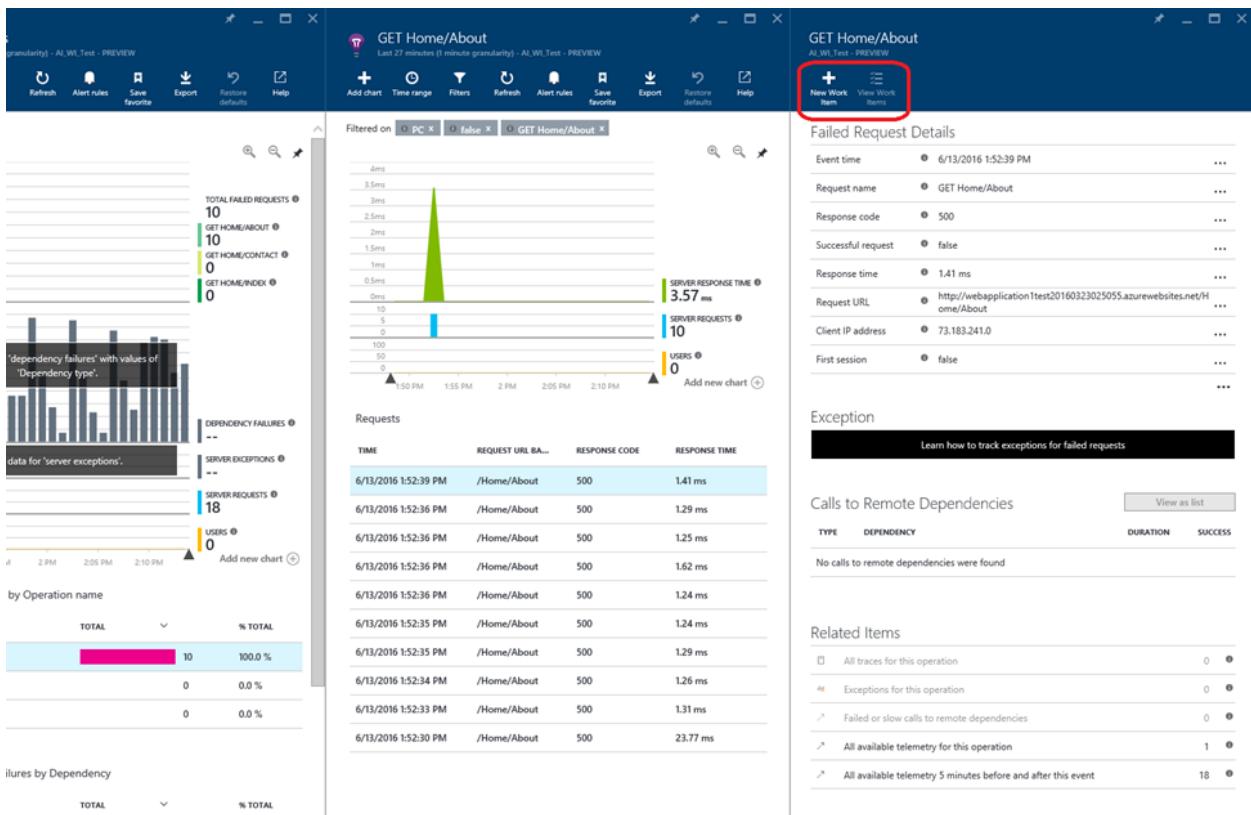
Creating work items from Application Insights is easy.

There are currently two locations where you can create work items: Proactive detection and individual instances of activity (that is, exceptions, failures, requests, and so on.). I'll show you a simple example, but the functionality is identical.

In this example, we're looking at a test web app we published to Azure. We started to drill into the activity for this app by looking at the Failures blade (but we could also get to this same information through the Search button or the Metrics Explorer):



We can see that I have several exceptions that fired when the user clicked on the Home/About tab on this web app. If I drill into this group of exceptions, I can see the list and then choose an individual exception:



Looking at the detailed blade for this exception, we see that there are now two buttons at the top that read "New Work Item" and "View Work Items."

To create a work item, I click on the first of these buttons, and it opens the new work item blade:

**GET Home/About**  
AI\_WI\_Test - PREVIEW

New Work Item   View Work Items

### Failed Request Details

|                    |   |     |
|--------------------|---|-----|
| Event time         | 6/13/2016 1:52:39 PM  | ... |
| Request name       | GET Home/About  | ... |
| Response code      | 500   | ... |
| Successful request | false   | ... |
| Response time      | 1.41 ms   | ... |
| Request URL        | http://webapplication1test20160323025055.azurewebsites.net/Home/About | ... |
| Client IP address  | 73.183.241.0  | ... |
| First session      | false   | ... |
|                    |   | ... |

### Exception

Learn how to track exceptions for failed requests

### Calls to Remote Dependencies

| TYPE                                       | DEPENDENCY | DURATION | SUCCESS |
|--|------------|----------|---------|
| No calls to remote dependencies were found |            |          |         |

### Related Items

|   |    |
|---|----|
| All traces for this operation                                 | 0  |
| Exceptions for this operation                                 | 0  |
| Failed or slow calls to remote dependencies                   | 0  |
| All available telemetry for this operation                    | 1  |
| All available telemetry 5 minutes before and after this event | 18 |

OK

As you can see, almost everything you need in your average scenario has been filled out for you.

The default values for "area path" and "assigned to" that you chose in the initial configuration are set, and all the detailed information we have available for this exception has been added to the details field.

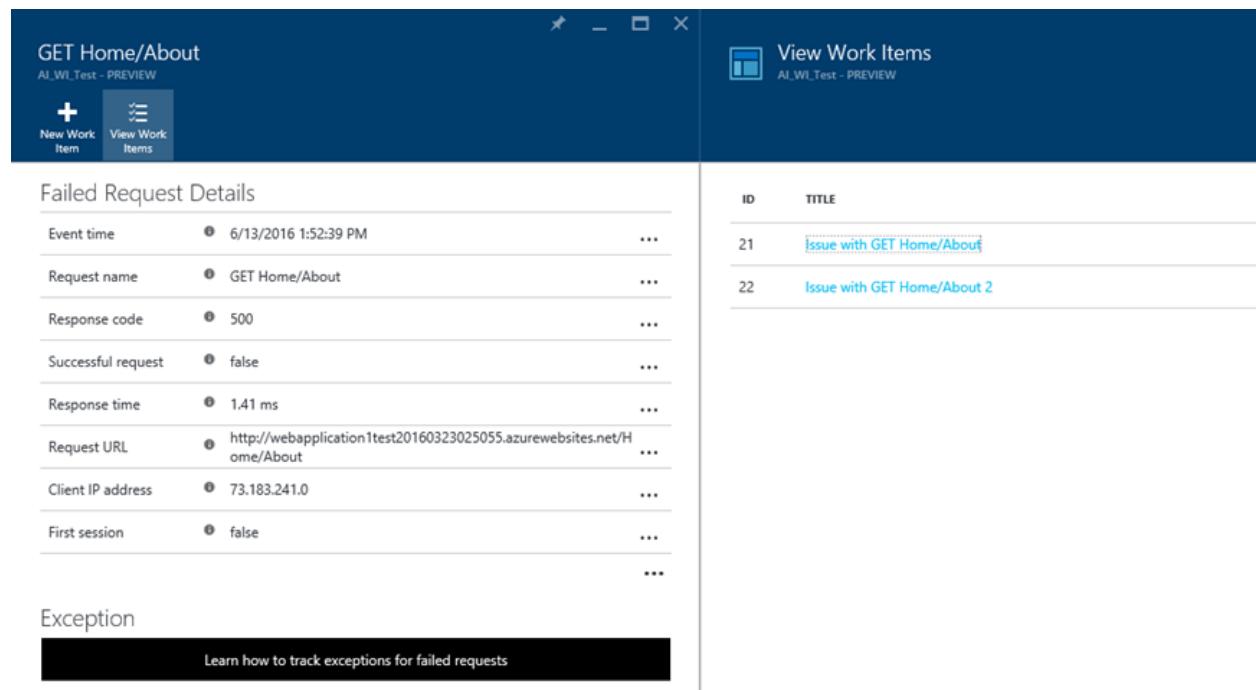
You can override the title and area path and assign them to fields in this blade, or you can add them to the captured details.

When you're ready to create your work item, click on the "OK" button, and your work item will be written to Azure DevOps.

## Viewing work items

Once you have created more work items in Application Insights, you can quickly view them in Azure DevOps.

The detailed blade for the event associated with the work item(s) will enable the "View Work Items" button if you are in the Azure portal. To see the list, click the button:



The screenshot shows two windows side-by-side. The left window is titled 'GET Home/About' and 'AI\_WI\_Test - PREVIEW'. It contains a 'New Work Item' button and a 'View Work Items' button. Below these are sections for 'Failed Request Details' and 'Exception'. The 'Failed Request Details' section lists various metrics: Event time (6/13/2016 1:52:39 PM), Request name (GET Home/About), Response code (500), Successful request (false), Response time (1.41 ms), Request URL (http://webapplication1test20160323025055.azurewebsites.net/Home/About), Client IP address (73.183.241.0), and First session (false). The right window is titled 'View Work Items' and 'AI\_WI\_Test - PREVIEW'. It shows a table with two work items: ID 21 (Title: Issue with GET Home/About) and ID 22 (Title: Issue with GET Home/About 2).

| ID | TITLE                                       |
|----|---|
| 21 | <a href="#">Issue with GET Home/About</a>   |
| 22 | <a href="#">Issue with GET Home/About 2</a> |

If you click the link for the work item that you want to view, it will open in Azure DevOps:

The screenshot shows the Azure DevOps interface with the 'WORK' tab selected. On the left, a sidebar lists 'Assigned to me', 'Recent work items' (with 'Bug 21' selected), 'My favorites', 'Team favorites', 'My Queries', and 'Shared Queries'. The main area displays a query result for 'BUG 21: Issue with GET Home/About'. The card shows the status as 'New' and 'Unassigned'. It includes fields for 'Area' (AI\_Test\_Playground) and 'Iteration' (AI\_Test\_Playground). A 'Details' section on the right shows 'Priority' (2), 'Severity' (3 - Medium), and 'Effort'. Below the card, sections for 'Repro Steps' and 'System Info' are visible.

## Advanced Configuration

Some of you may have noticed that there's a switch on the configuration blade that is labeled "Advanced Configuration."

We have provided another functionality to help you configure your ability to write to Azure DevOps in scenarios where you've changed or extended some out-of-the-box settings.

An excellent example of it is choosing more required fields. There's no way to handle this other-required mapping in the standard config, but you can handle it in advanced mode.

If you click on the switch, the controls at the bottom of the blade will change to look like this:

The screenshot shows a configuration page with the following sections:

- Authorization complete**: A status message with a **Required** link.
- Incomplete**: A button.
- Advanced Configuration**: A section with **On** and **Off** buttons.
- JSON**: A code editor containing the following JSON:

```
{  
    "/fields/System.AssignedTo": "",  
    "/fields/System.AreaPath": "Default AreaPath"  
}
```

Below the JSON editor, a note says: "By selecting OK, you are authorizing this resource to create or edit work items with the configured tracking system."

You can see that you're now given a JSON-based editing box where you can specify all the settings/mappings you might need to modify your Azure DevOps project.

## Next steps

We think it's an excellent start to integrating work item functionality with Application Insights.

But please remember that it's essentially the 1.0 version of this feature set.

We have much work planned, and you'll see a significant evolution in this space over the upcoming months.

Just for starters, let me outline a few of the things that we already have planned or are investigating:

- *Support for all work item types* – You probably noticed that the current feature set locks the work item type to just "bug." Logging bugs were our primary ask for this space, so that is where we started, but we certainly don't think that is where things should end. One of the more near-term changes you'll see is handling all work item types for all supported processes in Azure DevOps.
- *Links back to Application Insights* – It's great to create a work item with App Insights data in it, but what happens when you are in your ALM/DevOps system and looking at that item and want to quickly navigate back to the

source of the work item in App Insights? We plan to rapidly add links to the work items to make this as fast and easy as possible.

- *More flexible configuration* – Our standard configuration only handles scenarios where users haven't modified/extended their project in Azure DevOps. Today, if you have made these changes, you must switch to advanced configuration mode. In the future, we want to handle everyday things that people might change (for example, making more fields require or adding new fields) in the standard configuration wherever possible. It requires some updates from our friends on the Azure DevOps team, but they're already working on some of these for us. Once they're available, we'll make the standard configuration more flexible. In the meantime (and in the future), you can always use the advanced configuration to overcome limitations.
- *Multiple profiles* – Setting up a single configuration means that in shops where there are several ways users commonly create work items, the people creating work items from Application Insights would have to override values frequently. We plan to allow users to set up 1:n profiles, with common values specified for each, so that when you want to create a work item with that profile, you can choose it from a drop-down list.
- *More sources of creation for work items* – We'll continue to investigate (and take feedback on) other places in Application Insights where it makes sense to create work items.
- *Automatic creation of work items* – We can imagine scenarios where we might want a work item to be created for us based upon criteria. It is on the radar, but we're spending some design time to limit the possibilities of super-noisy or runaway work item creation. We believe this is a powerful and convenient feature, but we want to reduce the potential for spamming the ALM/DevOps system as much as possible.
- *Support for other ALM/DevOps systems* – We think Azure DevOps is an excellent product, but we recognize that many of our users may use some other product for their ALM/DevOps, and we want to meet people where they are. So, we're working on different first-tier integrations of popular ALM/DevOps products. We also plan to provide a pure custom configuration choice (like advanced config for Azure DevOps) so that end users will hook up Application Insights to virtually any ALM/DevOps system.

## Share team knowledge using Azure Project Wiki

Completed 100 XP

- 45 minutes

**Estimated time:** 45 minutes.

**Lab files:** none.

## Scenario

In this lab, you'll create and configure a wiki in Azure DevOps, including managing markdown content and creating a Mermaid diagram.

## Objectives

After completing this lab, you'll be able to:

- Create a wiki in an Azure Project.
- Add and edit markdown.
- Create a Mermaid diagram.

## Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).

## Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Publish code as a wiki.
- Exercise 2: Create and manage a project wiki.

# Develop monitor and status dashboards

## Explore Azure Dashboards

Completed 100 XP

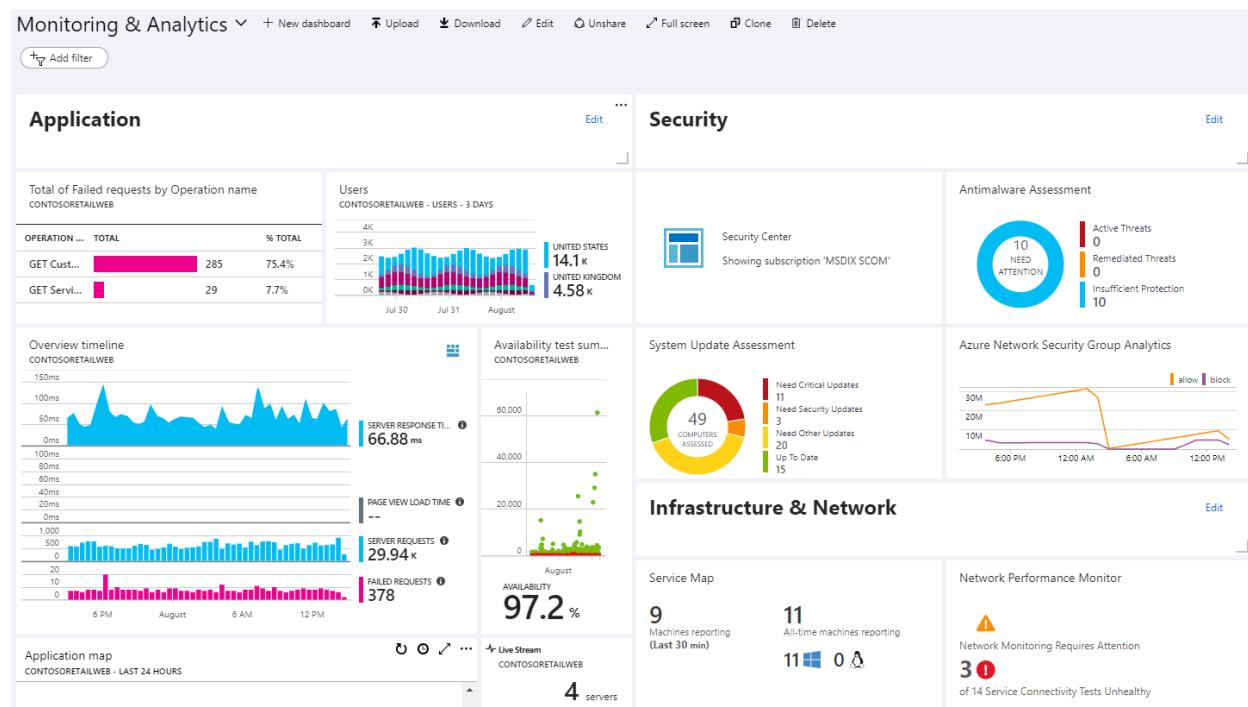
- 2 minutes

Visualizations such as charts and graphs can help you analyze your monitoring data to drill down on issues and identify patterns.

Depending on the tool you use, you may also share visualizations with other users inside and outside of your organization.

Azure dashboards are the primary dashboarding technology for Azure.

They're handy in providing a single pane of glass over your Azure infrastructure and services allowing you to identify critical issues quickly.



## Advantages

- Deep integration into Azure. Visualizations can be pinned to dashboards from multiple Azure pages, including metrics analytics, log analytics, and Application Insights.
- Supports both metrics and logs.
- Combine data from multiple sources, including output from [Metrics explorer](#), [Log Analytics queries](#), and [maps](#) and [availability](#) in Application Insights.
- Option for personal or shared dashboards. It's integrated with [Azure role-based authentication \(RBAC\)](#).
- Automatic refresh. Metrics refresh depends on the time range with a minimum of five minutes. Logs refresh at one minute.
- Parametrized metrics dashboards with timestamp and custom parameters.
- Flexible layout options.
- Full-screen mode.

## Limitations

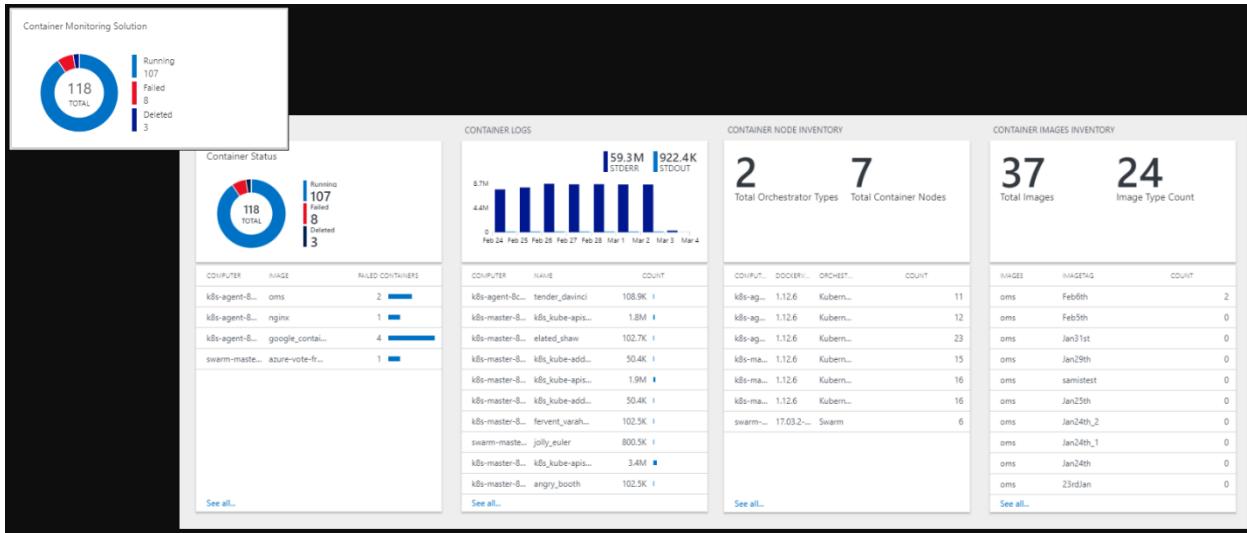
- Limited control over log visualizations with no support for data tables. The total number of data series is limited to 10, with different data series grouped under another bucket.
- No custom parameters support for log charts.
- Log charts are limited to the last 30 days.
- Log charts can only be pinned to shared dashboards.
- No interactivity with dashboard data.
- Limited contextual drill-down.

# Examine view designer in Azure Monitor

Completed 100 XP

- 1 minute

[View Designer in Azure Monitor](#) allows you to create custom visualizations with log data. They're used by [monitoring solutions](#) to present the data they collect.



## Advantages

- Rich visualizations for log data.
- Export and import views to transfer them to other resource groups and subscriptions.
- Integrates into Log Analytics management model with workspaces and monitoring solutions.
- Filters for custom parameters.
- Interactive supports multi-level drill-in (a view that drills into another view).

## Limitations

- Supports logs but not metrics.
- No personal views. Available to all users with access to the workspace.
- No automatic refresh.
- Limited layout options.
- No support for querying across multiple workspaces or Application Insights applications.
- Queries are limited in response size to 8 MB and query execution time of 110 seconds.

# Explore Azure Monitor workbooks

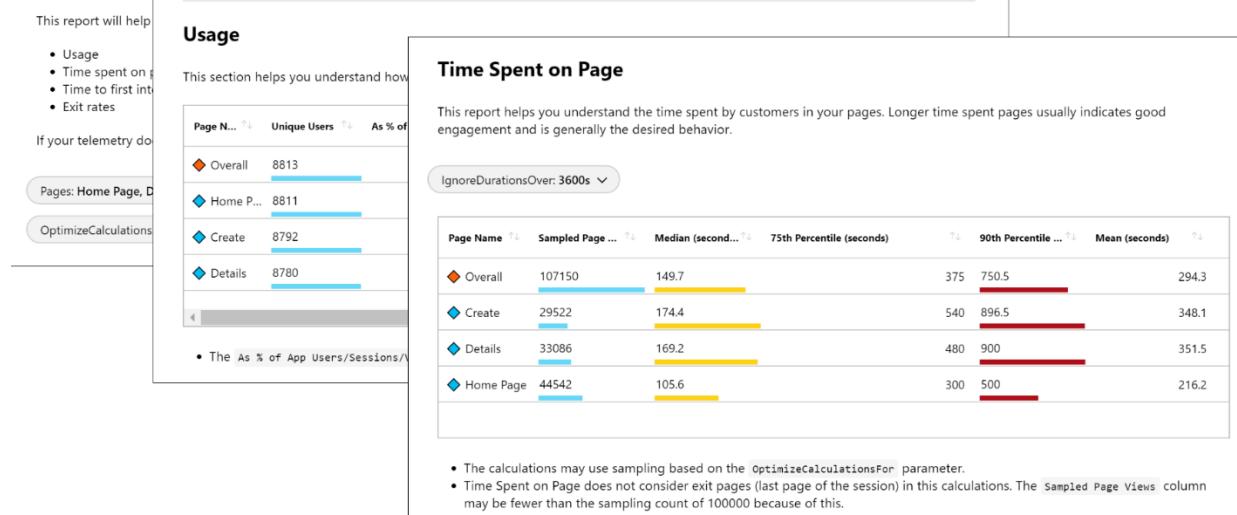
Completed 100 XP

- 1 minute

[Workbooks](#) are interactive documents that provide deep insights into your data, investigation, and collaboration inside the team. Specific examples where workbooks are helpful are troubleshooting guides and incident postmortem.

## Analysis of Page Views

Page views correspond to user activity in your app. Understanding how your users interact with your pages will give you good insights into what is working in your app and what aspects need improvements.



## Advantages

- Supports both metrics and logs.
- Supports parameters enabling interactive reports selecting an element in a table will dynamically update associated charts and visualizations.
- Document-like flow.
- Option for personal or shared workbooks.
- Easy, collaborative-friendly authoring experience.
- Templates support the public GitHub-based template gallery.

## Limitations

- No automatic refresh.
- No dense layout like dashboards, which make workbooks less useful as a single pane of glass. It's intended more for providing more profound insights.

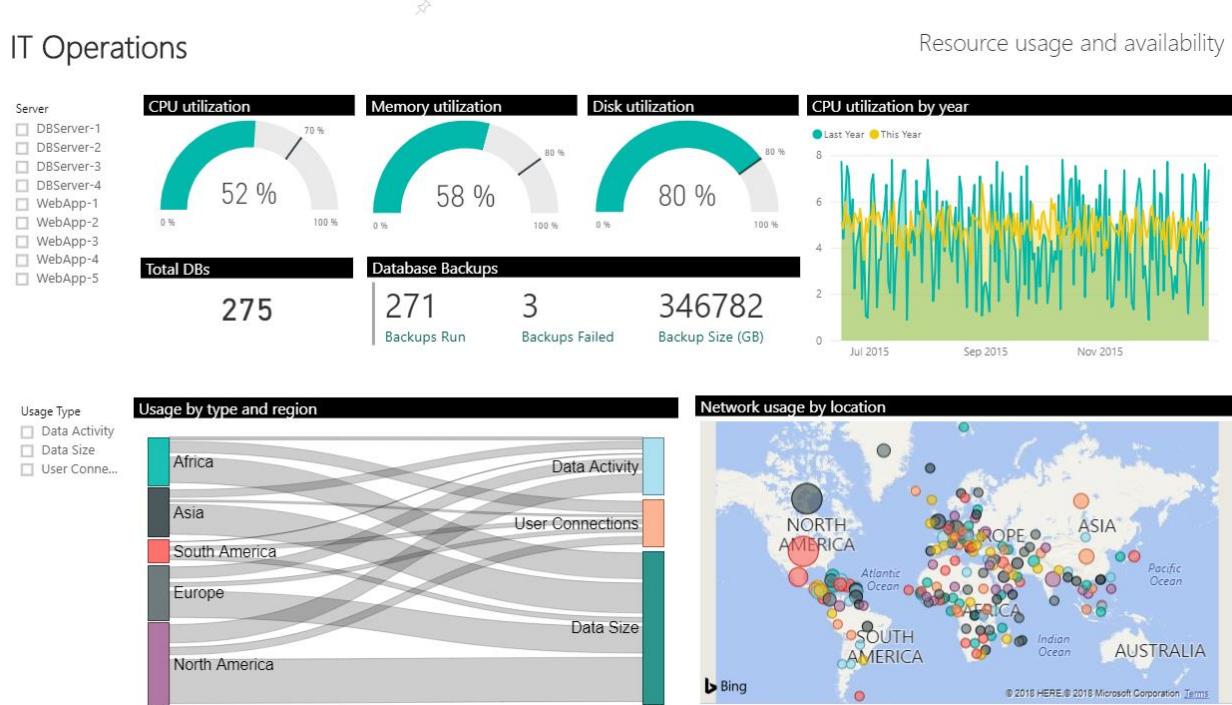
# Explore Power BI

Completed 100 XP

- 1 minute

[Power BI](#) is beneficial for creating business-centric dashboards and reports analyzing long-term KPI trends.

You can [import the results of a log query](#) into a Power BI dataset to take advantage of its features, such as combining data from different sources and sharing reports on the web and mobile devices.



## Advantages

- Rich visualizations.
- Extensive interactivity, including zoom-in and cross-filtering.
- Easy to share throughout your organization.
- Integration with other data from multiple data sources.
- Better performance with results cached in a cube.

## Limitations

- It supports logs but not metrics.
- No Azure RM integration. Can't manage dashboards and models through Azure Resource Manager.
- Need to import query results into the Power BI model to configure. Limitation on result size and refresh.

For more details about limitations, see [Limitations of Power BI Q&A - Power BI](#).

# Build your own custom application

Completed 100 XP

- 1 minute

You can access data in log and metric data in Azure Monitor through their API using any REST client.

It allows you to build your custom websites and applications.

## Advantages

- Complete flexibility in UI, visualization, interactivity, and features.
- Combine metrics and log data with other data sources.

## Disadvantages

- Significant engineering effort is required.

# Introduction to Secure DevOps

## Describe SQL injection attack

Completed 100 XP

- 1 minute

Choose your shell

Bash  
PowerShell  
Azure portal

SQL Injection is an attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application.

Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others.

Criminals may use it to gain unauthorized access to, delete, or alter your sensitive data: customer information, personal data, trade secrets, intellectual property, and more.

SQL Injection attacks are among the oldest, most prevalent, and most dangerous web application vulnerabilities.

The OWASP organization (Open Web Application Security Project) lists injections in their OWASP Top 10 2017 document as the number one threat to web application security.

## There's more

The Azure security center team has other [playbooks](#) you can look at to learn how vulnerabilities are exploited to trigger a virus attack and a DDoS attack.

## Understand DevSecOps

Completed 100 XP

- 1 minute

Choose your shell

BashPowerShellAzure portal

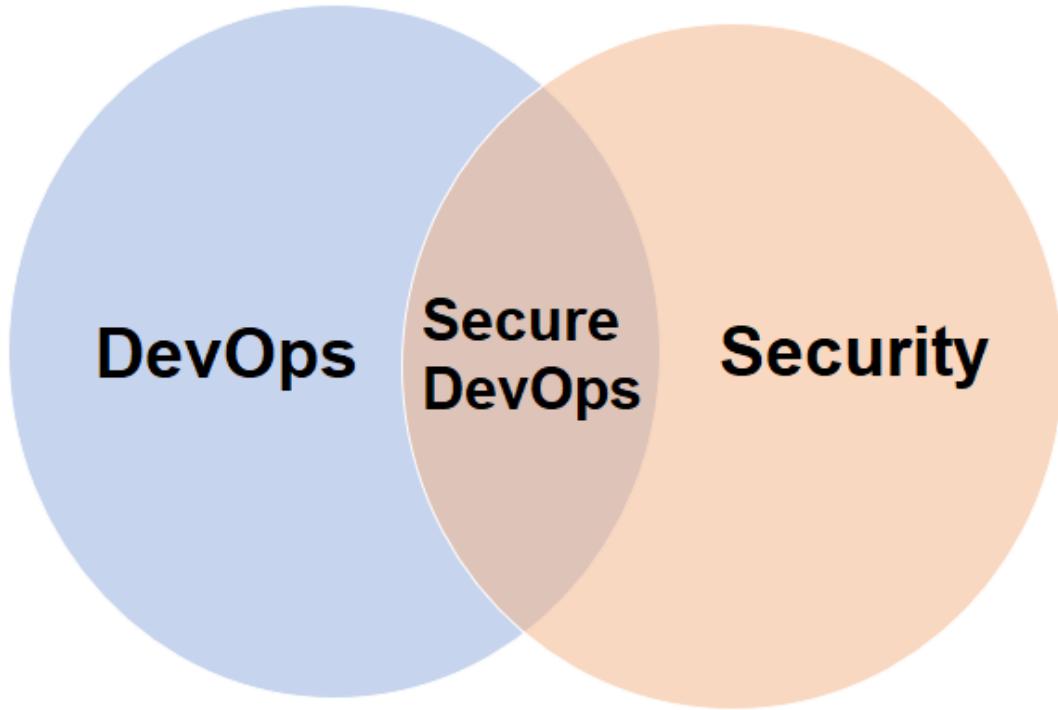
While adopting cloud computing is on the rise to support business productivity, a lack of security infrastructure can inadvertently compromise data.

The 2018 Microsoft Security Intelligence Report finds that:

- Data **isn't** encrypted both at rest and in transit by:
  - 7% of software as a service (SaaS) storage apps.
  - 86% percent of SaaS collaboration apps.
- HTTP headers session protection is supported by **only**:
  - 4% of SaaS storage apps.
  - 3% of SaaS collaboration apps.

## Secure DevOps (or DevSecOps)

*DevOps* is about working faster. *Security* is about-emphasizing thoroughness. Security concerns are typically addressed at the end of the cycle. It can potentially create unplanned work right at the end of the pipeline. *Secure DevOps* integrates DevOps with security into a set of practices designed to meet the goals of both DevOps and safety effectively.



A Secure DevOps pipeline allows development teams to work fast without breaking their project by introducing unwanted security vulnerabilities.

#### Note

Secure DevOps is also sometimes referred to as *DevSecOps*. You might encounter both terms, but each refers to the same concept.

## Security in the context of Secure DevOps

Historically, security typically operated on a slower cycle and involved traditional security methodologies, such as:

- Access control.
- Environment hardening.
- Perimeter protection.

Secure DevOps includes these traditional security methodologies and more. With Secure DevOps, security is about securing the pipeline.

Secure DevOps involves determining where to add protection to the elements that plug into your build and release pipelines.

Secure DevOps can show you how and where you can add security to your automation practices, production environments, and other pipeline elements while benefiting from the speed of DevOps.

Secure DevOps addresses broader questions, such as:

- Is my pipeline consuming third-party components, and are they secure?
- Are there known vulnerabilities within any of the third-party software we use?
- How quickly can I detect vulnerabilities (also called *time to detect*)?
- How quickly can I remediate identified vulnerabilities (also known as *time to remediate*)?

Security practices for detecting potential security anomalies must be as robust and fast as your DevOps pipeline's other parts. It also includes infrastructure automation and code development.

## Explore Secure DevOps Pipeline

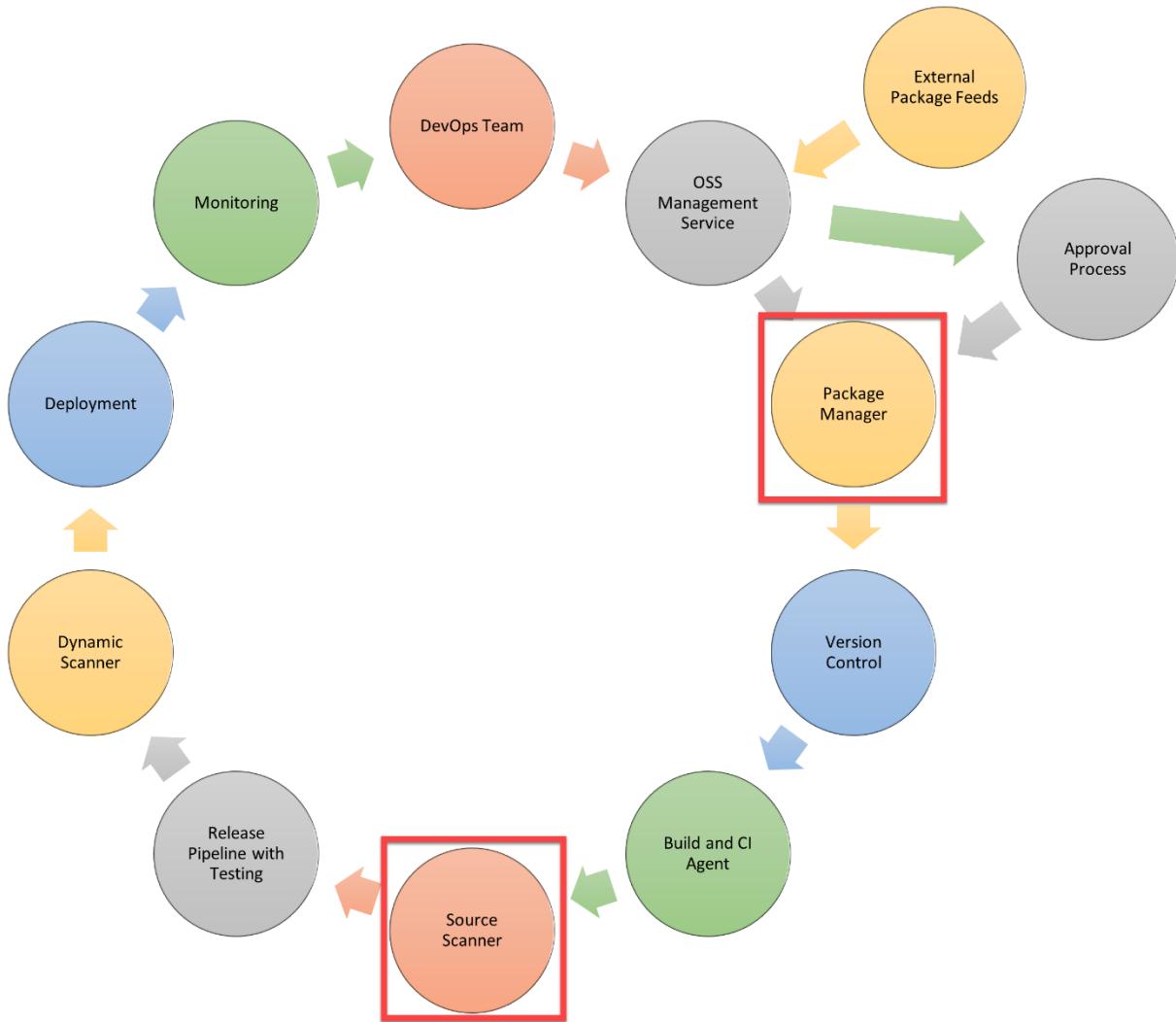
Completed 100 XP

- 1 minute

Choose your shell

BashPowerShellAzure portal

As previously stated, the goal of a *Secure DevOps Pipeline* is to enable development teams to work fast without introducing unwanted vulnerabilities to their project.



Two essential features of Secure DevOps Pipelines that aren't found in standard DevOps Pipelines are:

- Package management and the approval process associated with it. The previous workflow diagram details other steps for adding software packages to the Pipeline and the approval processes that packages must go through before they're used. These steps should be enacted early in the Pipeline to identify issues sooner in the cycle.
- Source Scanner is also an extra step for scanning the source code. This step allows for security scanning and checking for vulnerabilities that aren't present in the application code. The scanning occurs *after* the app is built *before* release and pre-release testing. Source scanning can identify security vulnerabilities earlier in the cycle.

In the rest of this lesson, we address these two essential features of Secure DevOps Pipelines, the problems they present, and some of the solutions for them.

## Explore key validation points

Completed 100 XP

- 1 minute

Choose your shell

BashPowerShellAzure portal

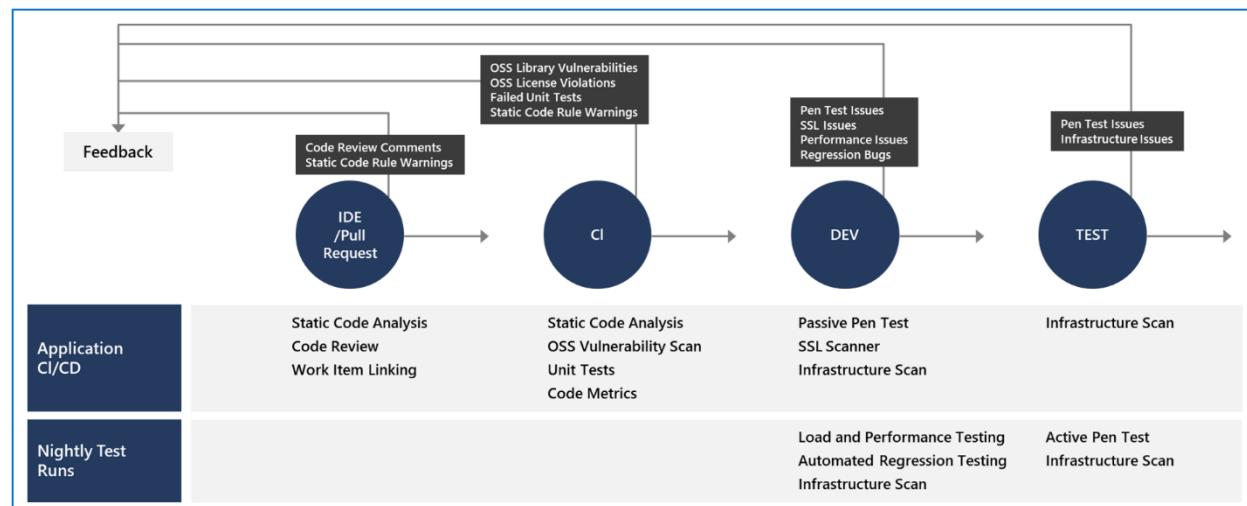
Continuous security validation should be added at each step from development through production to help ensure the application is always secure.

This approach aims to switch the conversation with the security team from approving each release to consenting to the CI/CD process and monitor and audit the process at any time.

The diagram below highlights the critical validation points in the CI/CD pipeline when building green field applications.

You may gradually implement the tools depending on your platform and your application's lifecycle.

Especially if your product is mature and you haven't previously run any security validation against your site or application.



## IDE / pull request

Validation in the CI/CD begins before the developer commits their code.

Static code analysis tools in the IDE provide the first line of defense to help ensure that security vulnerabilities aren't introduced into the CI/CD process.

The process for committing code into a central repository should have controls to help prevent security vulnerabilities from being introduced.

Using Git source control in Azure DevOps with branch policies provides a gated commit experience that can provide this validation.

Enabling branch policies on the shared branch requires a pull request to start the merge process and ensure the execution of all defined controls.

The pull request should require a code review, the one manual but important check for identifying new issues introduced into your code.

Along with this manual check, commits should be linked to work items for auditing why the code change was made and require a continuous integration (CI) build process to succeed before the push can be completed.

## Explore continuous security validation

Completed 100 XP

- 1 minute

Choose your shell

BashPowerShellAzure portal

Today, developers don't hesitate to use components available in public package sources (such as npm or NuGet).

With faster delivery and better productivity, open-source software (OSS) components are encouraged across many organizations.

However, as the dependency on these third-party OSS components increases, the risk of security vulnerabilities or hidden license requirements also increases compliance issues.

For a business, it's critical, as issues related to compliance, liabilities, and customer personal data can cause many privacy and security concerns.

Identifying such issues early in the release cycle gives you an advanced warning and enough time to fix the problems. Notably, the cost of rectifying issues is lower the earlier the project discovers the problem.

Many tools can scan for these vulnerabilities within the build and release pipelines.

Once the merge is complete, the CI build should execute as part of the pull request (PR-CI) process.

Typically, the primary difference between the two runs is that the PR-CI process doesn't need any packaging/staging in the CI build.

These CI builds should run static code analysis tests to ensure that the code follows all rules for both maintenance and security.

Several tools can be used for it:

- SonarQube.
- Visual Studio Code Analysis and the Roslyn Security Analyzers.
- Checkmarx - A Static Application Security Testing (SAST) tool.
- BinSkim - A binary static analysis tool that provides security and correctness results for Windows portable executables and many more.

Many of the tools seamlessly integrate into the Azure Pipelines build process. Visit the Visual Studio Marketplace for more information on the integration capabilities of these tools.

Also, to verify code quality with the CI build, two other tedious or ignored validations are scanning third-party packages for vulnerabilities and OSS license usage.

The response is fear or uncertainty when we ask about third-party package vulnerabilities and licenses.

Organizations trying to manage third-party packages vulnerabilities or OSS licenses explain that their process is tedious and manual.

Fortunately, Mend Software's tools can make this identification process almost instantaneous.

In a later module, we'll discuss integrating several helpful and commonly used security and compliance tools.

# Understand threat modeling

Completed 100 XP

- 1 minute

Choose your shell

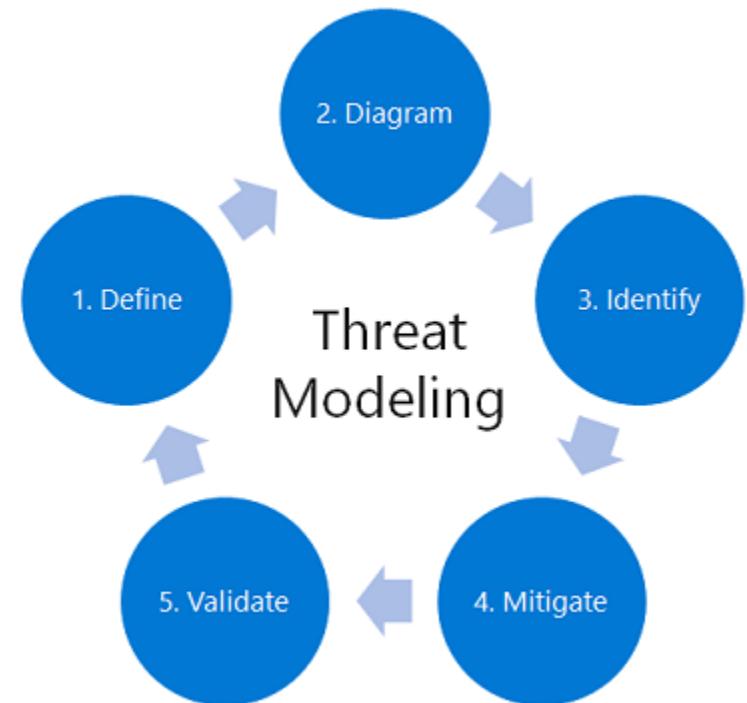
BashPowerShellAzure portal

Threat modeling is a core element of the Microsoft Security Development Lifecycle (SDL).

It's an engineering technique you can use to help you identify threats, attacks, vulnerabilities, and countermeasures that could affect your application.

You can use threat modeling to shape your application's design, meet your company's security goals, and reduce risk.

With non-security experts in mind, the tool makes threat modeling easier for all developers by providing clear guidance on creating and analyzing threat models.



There are five major threat modeling steps:

- Defining security requirements.
- Creating an application diagram.
- Identifying threats.
- Mitigating threats.
- Validating that threats have been mitigated.

Threat modeling should be part of your typical development lifecycle, enabling you to refine your threat model and progressively reduce risk.

## Microsoft Threat Modeling Tool

The Microsoft Threat Modeling Tool makes threat modeling easier for all developers through a standard notation for visualizing system components, data flows, and security boundaries.

It also helps threat modelers identify classes of threats they should consider based on the structure of their software design.

The tool has been designed with non-security experts in mind, making threat modeling easier for all developers by providing clear guidance on creating and analyzing threat models.

The Threat Modeling Tool enables any developer or software architect to:

- Communicate about the security design of their systems.
- Analyze those designs for potential security issues using a proven methodology.
- Suggest and manage mitigation for security issues.

For more information, you can see:

- [Threat Modeling Tool feature overview](#).
- [Microsoft Threat Modeling Tool](#).

## Exercise threat modeling

Completed 100 XP

- 1 minute

Choose your shell

BashPowerShellAzure portal

Security can't be a separate department in a silo. It also can't be added at the end of a project. Security must be part of DevOps, and together they're called DevSecOps.

The biggest weakness is not knowing the flaw in your solution. Microsoft has created a threat modeling tool to remediate it, which helps you understand potential security vulnerabilities in your solution.

The Threat Modeling Tool is a core element of the Microsoft Security Development Life cycle (SDL).

It allows software architects to identify and mitigate potential security issues early when they're relatively easy and cost-effective to resolve.

As a result, it dramatically reduces the total cost of development.

The tool has been designed with non-security experts in mind, making threat modeling easier for all developers by providing clear guidance on creating and analyzing threat models.

The tool enables anyone to:

- Communicate about the security design of their systems.
- Analyze those designs for potential security issues using a proven methodology.
- Suggest and manage mitigations for security issues.

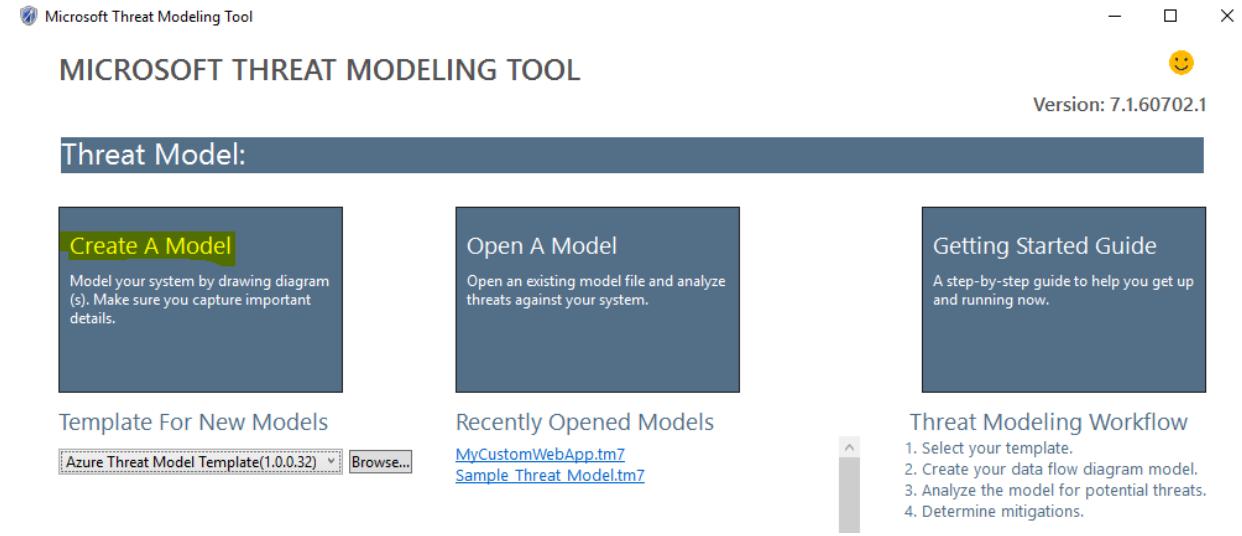
In this exercise, we'll see how easy it's to use the Threat Modeling tool to see potential vulnerabilities in your infrastructure solution that one should consider when provisioning and deploying the Azure resources and the application solution into the solution.

## Getting started

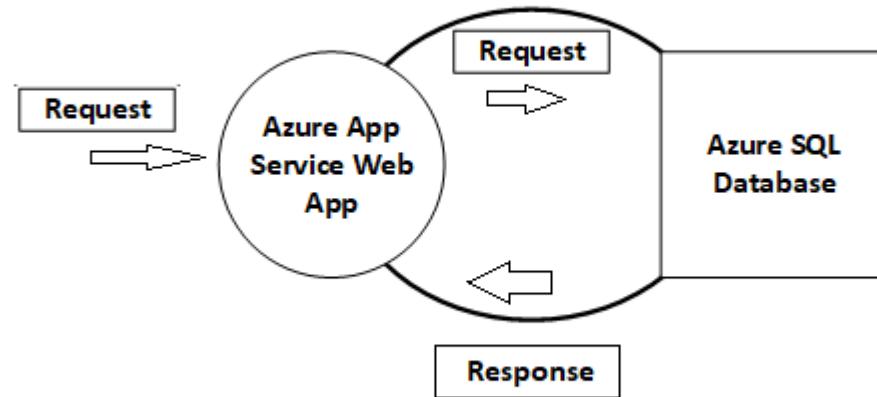
- Download and install the [Threat Modeling tool](#).

## How to do it

1. Launch the Microsoft Threat Modeling Tool and choose the option to Create a Model.



- From the right panel, search and add Azure App Service Web App and Azure SQL Database, and link them up to show a request and response flow, as demonstrated in the following image.



- From the toolbar menu, select View -> Analysis view. The analysis view will show you a complete list of threats categorized by severity.

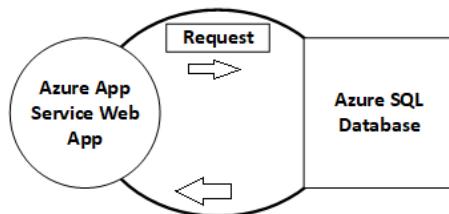
| Threat List  |  |             |   |  |
|--|--|-------------|---|--|
| Short Description  | Description  | Interaction | Possible Mitigation   |  |
| A user subject gains increased capability or privilege by t... | Due to poorly configured account policies, adversary can gain unauthorized access to Azure SQL Database. | Request     | When possible use Azure Active Directory Authentication for connecting to SQL Database. Refer: <a href="https://aka.ms/tmt-th10a">https://aka.ms/tmt-th10a</a>  |  |
| A user subject gains increased capability or privilege by t... | An adversary can gain unauthorized access to Azure SQL Database due to weak account policy.              | Request     | Restrict access to Azure SQL Database instances by configuring server-level and database-level firewall rules to permit connections from specific IP addresses or a custom set of IP addresses where possible. Refer: <a href="https://aka.ms/tmt-th10b">https://aka.ms/tmt-th10b</a> and <a href="https://aka.ms/tmt-th10c">https://aka.ms/tmt-th10c</a> |  |
| Information disclosure happens when the information ca...      | An adversary can read confidential data due to weak network security configuration.                      | Request     | Enable Transport Layer Security (TLS) encryption for all communication between the application and the database.  |  |
| Information disclosure happens when the information ca...      | An adversary having access to the storage container can read confidential data.                          | Request     | Enable SQL Database Auditing to log all database activity and alert on suspicious activity.   |  |
| A user subject gains increased capability or privilege by t... | A compromised identity may permit more privileges than intended.   | Request     | It is recommended to use multi-factor authentication for all accounts.  |  |
| Repudiation threats involve an adversary denying that so...    | An adversary can deny actions performed on Azure SQL Database.   | Request     | Enable audit logs to capture all database activity and review them for suspicious activity.   |  |
| A user subject gains increased capability or privilege by t... | An adversary can gain long term, persistent access to the database.                                      | Request     | It is recommended to use long-term, secure credentials for database connections.  |  |
| A user subject gains increased capability or privilege by t... | An adversary may abuse weak Azure SQL Database account policies to gain access.                          | Request     | Enable SQL Database Auditing to log all database activity and alert on suspicious activity.   |  |
| Denial of Service happens when the process or a datasto...     | An adversary may block access to the application or database.  | Response    | Network level firewalls and load balancers can help mitigate Denial of Service attacks.   |  |
| A user subject gains increased capability or privilege by t... | An adversary may gain long term persistent access to the database.                                       | Response    | Store secrets securely and use strong, unique passwords for database accounts.  |  |
| A user subject gains increased capability or privilege by t... | An adversary may gain unauthorized access to Azure SQL Database.   | Response    | Restrict access to Azure SQL Database instances by configuring server-level and database-level firewall rules to permit connections from specific IP addresses or a custom set of IP addresses where possible. Refer: <a href="https://aka.ms/tmt-th10b">https://aka.ms/tmt-th10b</a> and <a href="https://aka.ms/tmt-th10c">https://aka.ms/tmt-th10c</a> |  |

[Export Csv](#)

12 Threats Displayed, 12 Total

- To generate a full report of the threats, select Reports -> Create a full report from the toolbar menu, and select a location to save the report.

A full report is generated with details of the threat, the SLDC phase it applies to, possible mitigation, and links to more information.



1. An adversary can gain unauthorized access to Azure SQL database due to weak account policy [State: Not Started] [Priority: High]

Category: Elevation of Privileges

Description: Due to poorly configured account policies, adversary can launch brute force attacks on Azure SQL Database

Justification: <no mitigation provided>

Possible Mitigation(s): When possible use Azure Active Directory Authentication for connecting to SQL Database. Refer: <https://aka.ms/tmt-th10a> and <https://aka.ms/tmt-th10b> to connect to Database server. Refer: <https://aka.ms/tmt-th10c>

SDL Phase: Implementation

2. An adversary can gain unauthorized access to Azure SQL DB instances due to weak network security configuration. [State: Not Started]

Category: Elevation of Privileges

Description: An adversary can gain unauthorized access to Azure SQL DB instances due to weak network security configuration.

Justification: <no mitigation provided>

Possible Mitigation(s): Restrict access to Azure SQL Database instances by configuring server-level and database-level firewall rules to permit connections from specific IP addresses or a custom set of IP addresses where possible. Refer:<https://aka.ms/tmt-th10b> and <https://aka.ms/tmt-th10c>

SDL Phase: Implementation

# Security Monitoring and Governance

## Implement pipeline security

Completed 100 XP

- 2 minutes

It's fundamental to protect your code protecting credentials, and secrets. Phishing is becoming ever more sophisticated. The following list is several operational practices that a team ought to apply to protect itself:

- Authentication and authorization. Use multifactor authentication (MFA), even across internal domains, and just-in-time administration tools such as Azure PowerShell [Just Enough Administration \(JEA\)](#), to protect against privilege escalations. Using different passwords for different user accounts will limit the damage if a set of access credentials is stolen.
- The CI/CD Release Pipeline. If the release pipeline and cadence are damaged, use this pipeline to rebuild infrastructure. Manage Infrastructure as Code (IaC) with Azure Resource Manager or use the Azure platform as a service (PaaS) or a similar service. Your pipeline will automatically create new instances and then destroy them. It limits the places where attackers can hide malicious code inside your infrastructure. Azure DevOps will encrypt the secrets in your pipeline. As a best practice, rotate the passwords just as you would with other credentials.
- Permissions management. You can manage permissions to secure the pipeline with role-based access control (RBAC), just as you would for your source code. It keeps you in control of editing the build and releases definitions that you use for production.
- Dynamic scanning. It's the process of testing the running application with known attack patterns. You could implement penetration testing as part of your release. You also could keep up to date on security projects such as the Open Web Application Security Project ([OWASP](#)) Foundation, then adopt these projects into your processes.
- Production monitoring. It's a critical DevOps practice. The specialized services for detecting anomalies related to intrusion are known as *Security Information and Event Management*. [Microsoft Defender for Cloud](#) focuses on the security incidents related to the Azure cloud.

### Note

In all cases, use Azure Resource Manager Templates or other code-based configurations. Implement IaC best practices, such as making changes in templates to make changes traceable and repeatable. Also, you can use provisioning and configuration technologies

such as Desired State Configuration (DSC), Azure Automation, and other third-party tools and products that can integrate seamlessly with Azure.

# Explore Microsoft Defender for Cloud

Completed 100 XP

- 2 minutes

**Microsoft Defender for Cloud** is a monitoring service that provides threat protection across all your services both in Azure and on-premises. Microsoft Defender can:

- Provide security recommendations based on your configurations, resources, and networks.
- Monitor security settings across on-premises and cloud workloads and automatically apply required security to new services as they come online.
- Continuously monitor all your services and do automatic security assessments to identify potential vulnerabilities before they can be exploited.
- Use Azure Machine Learning to detect and block malicious software from being installed on your virtual machines (VMs) and services. You can also define a list of allowed applications to ensure that only the validated apps can execute.
- Analyze and identify potential inbound attacks and help investigate threats and any post-breach activity that might have occurred.
- Provide just-in-time (JIT) access control for ports by reducing your attack surface by ensuring the network only allows traffic that you require.



Microsoft Defender for Cloud is part of the [Center for Internet Security \(CIS\) Benchmarks](#) recommendations.

## Microsoft Defender for Cloud versions

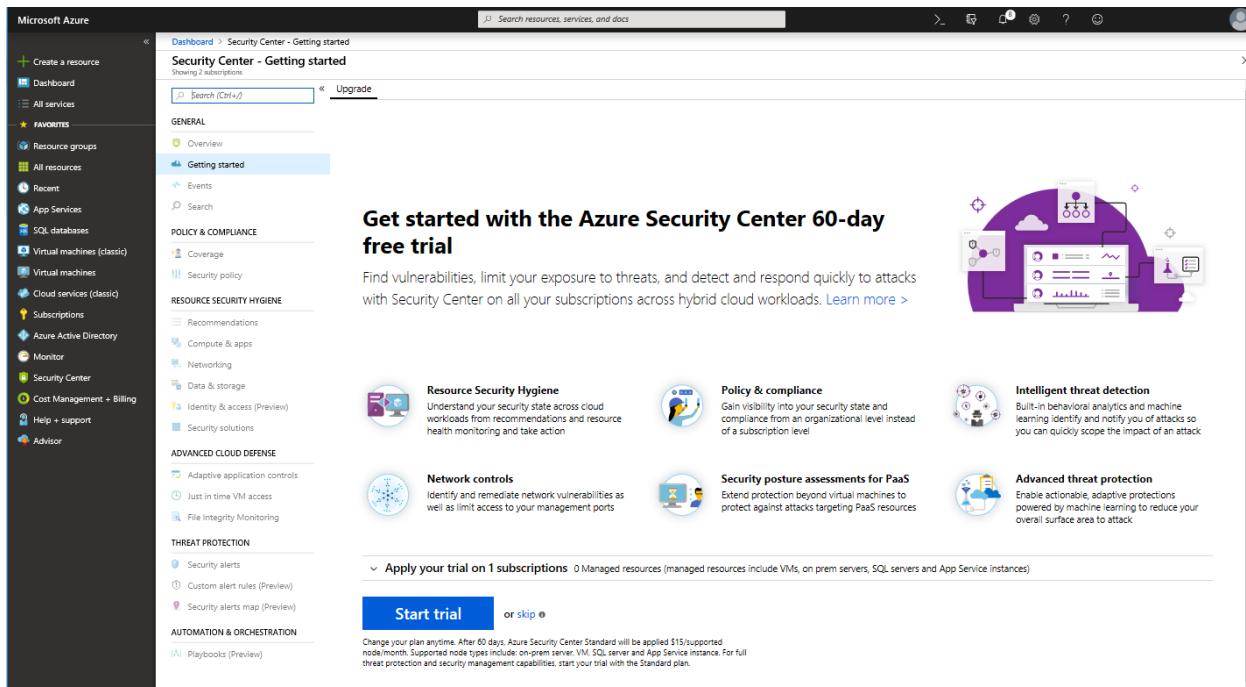
Microsoft Defender for Cloud supports both Windows and Linux operating systems. It can also provide security to features in IaaS and platform as a service (PaaS) scenarios.

Microsoft Defender for Cloud is available in two versions:

- Free. Available as part of your Azure subscription, this tier is limited to assessments and Azure resources' recommendations only.
- Standard. This tier provides a full suite of security-related services, including continuous monitoring, threat detection, JIT access control for ports, and more.

To access the full suite of Microsoft Defender for Cloud services, you'll need to upgrade to a Standard version subscription.

You can access the 60-day free trial from the Microsoft Defender for Cloud dashboard in the Azure portal.



You can read more about Microsoft Defender for Cloud at [Microsoft Defender for Cloud](#).

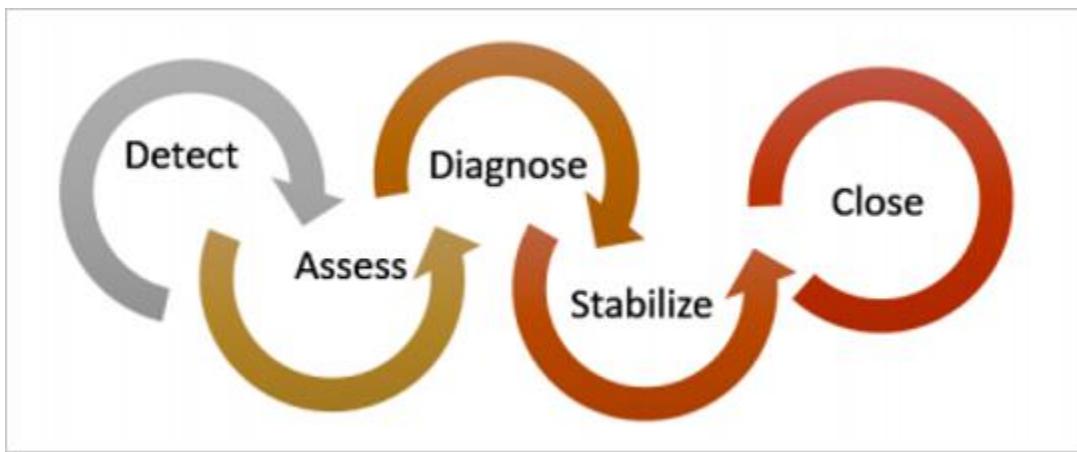
## Examine Microsoft Defender for Cloud usage scenarios

Completed 100 XP

- 2 minutes

You can integrate Microsoft Defender for Cloud into your workflows and use it in many ways. For example, use Microsoft Defender for Cloud as part of your incident response plan.

Many organizations only respond to security incidents after an attack has occurred. To reduce costs and damage, it's necessary to have an incident response plan *before* an attack occurs.



The following examples show how to use Microsoft Defender for Cloud to detect, assess, and diagnose your incident response plan stages.

- **Detect.** Review the first indication of an event investigation. For example, use the Microsoft Defender for Cloud dashboard to review a high-priority security alert's initial verification.
- **Assess.** Do the initial assessment to obtain more information about suspicious activity. For example, you can get more information about a security alert from Microsoft Defender for Cloud.
- **Diagnose.** Conduct a technical investigation and identify containment, mitigation, and workaround strategies. For example, you can follow the remediation steps described by Microsoft Defender for Cloud for a particular security alert.
- **Use Microsoft Defender for Cloud recommendations to enhance security.**

You can reduce the chances of a significant security event by configuring a security policy and then implementing the recommendations provided by Microsoft Defender for Cloud. A *security policy* defines the controls recommended for resources within a specified subscription or resource group. You can define policies in Microsoft Defender for Cloud according to your company's security requirements.

Microsoft Defender for Cloud analyzes the security state of your Azure resources. When identifying potential security vulnerabilities, it creates recommendations based on the controls set in the security policy.

The suggestions guide you through the process of configuring the corresponding security controls.

For example, if you have workloads that don't require the Azure SQL Database Transparent Data Encryption (TDE) policy, turn off the policy at the subscription level and enable it only on the resource groups where SQL Database TDE is required.

You can read more about the Microsoft Defender for Cloud at the [Microsoft Defender for Cloud](#). More implementation and scenario details are available in the [Microsoft Defender for Cloud planning and operations guide](#).

## Explore Azure Policy

Completed 100 XP

- 1 minute

**Azure Policy** is an Azure service that you can create, assign, and manage policies.

Policies enforce different rules and effects over your Azure resources, ensuring that your resources stay compliant with your standards and SLAs.



Azure Policy uses policies and initiatives to provide policy enforcement capabilities.

Azure Policy evaluates your resources by scanning for resources that don't follow the policies you create.

For example, you might have a policy that specifies a maximum size limit for VMs in your environment.

After you implement your maximum VM size policy, Azure Policy will evaluate the VM resource whenever a VM is created—or updated to ensure that the VM follows the size limit you set in your Policy.

Azure Policy can help maintain the state of your resources by evaluating your existing resources and configurations and automatically remediating non-compliant resources.

It has built-in policy and initiative definitions for you to use. The definitions are arranged into categories: Storage, Networking, Compute, Security Center, and Monitoring.

Azure Policy can also integrate with Azure DevOps by applying any continuous integration (CI) and continuous delivery (CD) pipeline policies that apply to the pre-deployment and post-deployment of your applications.

## CI/CD pipeline integration

The Check Gate task is an example of an Azure policy that you can integrate with your DevOps CI/CD pipeline.

Using Azure policies, Check gate provides security and compliance assessment on the resources with an Azure resource group or subscription that you can specify.

Check gate is available as a Release pipeline deployment task.

For more information, go to:

- [Azure Policy Check Gate task](#).
- [Azure Policy](#).

## Understand policies

Completed 100 XP

- 2 minutes

Applying a policy to your resources with Azure Policy involves the following high-level steps:

- Policy definition. Create a policy definition.
- Policy assignment. Assign the definition to a scope of resources.
- Remediation. Review the policy evaluation results and address any non-compliances.

## Policy definition

A **policy definition** specifies the resources to be evaluated and the actions to take on them. For example, you could prevent VMs from deploying if exposed to a public IP address. You could also contain a specific hard disk for deploying VMs to control costs. Policies are defined in the JavaScript Object Notation (JSON) format.

The following example defines a policy that limits where you can deploy resources:

JSONCopy

```
{
  "properties": {
    "mode": "all",
    "parameters": {
      "allowedLocations": {
        "type": "array",
        "metadata": {
          "description": "The list of locations that can be specified when deploying resources",
          "strongType": "location",
          "displayName": "Allowed locations"
        }
      },
      "displayName": "Allowed locations",
      "description": "This policy enables you to restrict the locations your organization can specify when deploying resources.",
      "policyRule": {
        "if": {
          "not": {
            "field": "location",
            "in": "[parameters('allowedLocations')]"
          }
        },
        "then": {
          "effect": "deny"
        }
      }
    }
  }
}
```

The following list is example policy definitions:

- Allowed Storage Account SKUs (Deny): Determines if a storage account being deployed is within a set of SKU sizes. Its effect is to deny all storage accounts that don't adhere to the set of defined SKU sizes.
- Allowed Resource Type (Deny): Defines the resource types that you can deploy. Its effect is to deny all resources that aren't part of this defined list.
- Allowed Locations (Deny): Restricts the available locations for new resources. Its effect is used to enforce your geo-compliance requirements.
- Allowed Virtual Machine SKUs (Deny): Specify a set of virtual machine SKUs you can deploy.
- Add a tag to resources (Modify): Applies a required tag and its default value if the deploy request does not specify it.
- Not allowed resource types (Deny): Prevents a list of resource types from being deployed.

## Policy assignment

Policy definitions, whether custom or built-in, need to be assigned.

A *policy assignment* is a policy definition that has been assigned to a specific scope. Scopes can range from a management group to a resource group.

Child resources will inherit any policy assignments applied to their parents.

It means if a policy is applied to a resource group, it's used to all the resources within that resource group.

However, you can define subscopes for excluding resources from policy assignments.

You can assign policies via:

- Azure portal.
- Azure CLI.
- PowerShell.

## Remediation

Resources found not to follow a *deployIfNotExists* or *modify* policy condition can be put into a compliant state through Remediation.

*Remediation* instructs Azure Policy to run the *deployIfNotExists* effect or the tag operations of the policy on existing resources.

To minimize configuration drift, you can bring resources into compliance using automated bulk Remediation instead of going through them one at a time.

You can read more about Azure Policy on the [Azure Policy](#) webpage.

## Explore initiatives

Completed 100 XP

- 1 minute

Initiatives work alongside policies in Azure Policy. An *initiative definition* is a set of policy definitions to help track your compliance state for meeting large-scale compliance goals.

Even if you have a single policy, we recommend using initiatives if you anticipate increasing your number of policies over time.

Applying an initiative definition to a specific scope is called an *initiative assignment*.

## Initiative definitions

Initiative definitions simplify the process of managing and assigning policy definitions by grouping sets of policies into a single item.

For example, you can create an initiative named *Enable Monitoring in Azure Security Center* to monitor security recommendations from Azure Security Center.

Under this example initiative, you would have the following policy definitions:

- Monitor unencrypted SQL Database in Security Center. This policy definition monitors unencrypted SQL databases and servers.
- Monitor OS vulnerabilities in Security Center. This policy definition monitors servers that don't satisfy a specified OS baseline configuration.
- Monitor missing Endpoint Protection in Security Center. This policy definition monitors servers without an endpoint protection agent installed.

## Initiative assignments

Like a policy assignment, an *initiative assignment* is an initiative definition assigned to a specific scope.

Initiative assignments reduce the need to make several initiative definitions for each scope.

Scopes can range from a management group to a resource group. You can assign initiatives in the same way that you assign policies.

You can read more about policy definition and structure at [Azure Policy definition structure](#).

## Explore resource locks

Completed 100 XP

- 1 minute

**Locks** help you prevent accidental deletion or modification of your Azure resources. You can manage locks from within the Azure portal.

In the Azure portal, locks are called **Delete** and **Read-only**, respectively.

Go to the Settings section on the resource's settings blade to review, add, or delete locks for a resource in the Azure portal.

You might need to lock a subscription, resource group, or resource to prevent users from accidentally deleting or modifying critical resources.

You can set a lock level to **CanNotDelete** or **ReadOnly**:

- **CanNotDelete** means that authorized users can read and modify a resource, but they can't delete it.
- **ReadOnly** means that authorized users can read a resource, but they can't modify or delete it.

You can read more about Locks on the [Lock resources to prevent unexpected changes](#) webpage.

# Explore Azure Blueprints

Completed 100 XP

- 2 minutes

Azure Blueprints enables cloud architects to define a repeatable set of Azure resources that implement and adhere to an organization's standards, patterns, and requirements.

Azure Blueprints helps development teams build and deploy new environments rapidly with a set of built-in components that speed up development and delivery.

Furthermore, it's done while staying within organizational compliance requirements.



Azure Blueprints provides a declarative way to orchestrate deployment for various resource templates and artifacts, including:

- Role assignments
- Policy assignments
- Azure Resource Manager templates
- Resource groups

To implement Azure Blueprints, complete the following high-level steps:

1. Create a blueprint.
2. Assign the blueprint.
3. Track the blueprint assignments.

With Azure Blueprints, the relationship between the blueprint definition (what *should be* deployed) and the blueprint assignment (what *is* deployed) is preserved.

The blueprints in Azure Blueprints are different from Azure Resource Manager templates.

When Azure Resource Manager templates deploy resources, they have no active relationship with the deployed resources. They exist in a local environment or source control.

By contrast, with Azure Blueprints, each deployment is tied to an Azure Blueprints package. It means that the relationship with resources will be maintained, even after deployment. This way, keeping relationships improves deployment tracking and auditing capabilities.

## Usage scenario

Whether government, industry, or organizational, Adhering to security and compliance requirements can be difficult and time-consuming.

To help you to trace your deployments and audit them for compliance, Azure Blueprints uses artifacts and tools that speed up your path to certification.

Azure Blueprints is also helpful in Azure DevOps scenarios. Mainly where blueprints are associated with specific build artifacts and release pipelines, and blueprints can be tracked rigorously.

You can learn more about Azure Blueprints at [Azure Blueprints](#).

## Understand Microsoft Defender for Identity

Completed 100 XP

- 2 minutes

**Microsoft Defender for Identity** (formerly Azure Advanced Threat Protection, also known as Azure ATP) is a cloud-based security solution that identifies and detects:

- Advanced threats.
- Compromised identities.
- Malicious insider actions directed at your organization.

Microsoft Defender can detect known malicious attacks and techniques and help you investigate security issues and network vulnerabilities.

# Microsoft Defender components

Microsoft Defender consists of the following components:

- Microsoft Defender portal. Microsoft Defender has its portal. You can monitor and respond to suspicious activity through the Microsoft Defender portal. The Microsoft Defender portal allows you to manage your Microsoft Defender instance and review data received from Microsoft Defender sensors.

You can also use the Microsoft Defender portal to monitor, manage, and investigate threats to your network environment.

You can sign into the Microsoft Defender portal at <https://security.microsoft.com/>.

Sign in with a user account assigned to a Microsoft Entra security group with access to the Microsoft Defender portal.

- Microsoft Defender sensor. Microsoft Defender sensors are installed directly on your domain controllers. The sensors monitor domain controller traffic without requiring a dedicated server or port mirroring configurations.
- Microsoft Defender cloud service. The Microsoft Defender cloud service runs on the Azure infrastructure and is deployed in the United States, Europe, and Asia. The Microsoft Defender cloud service is connected to the Microsoft Intelligent Security Graph.

The screenshot shows the Microsoft Defender for Cloud Overview page in the Azure portal. The top navigation bar includes 'Microsoft Azure (Preview)', 'Report a bug', and a search bar. The main header says 'Microsoft Defender for Cloud | Overview' with a note 'Showing 9 subscriptions'. On the left, there's a sidebar with sections like 'General' (Overview, Getting started, Recommendations, Security alerts, Inventory, Workbooks, Community, Diagnose and solve problems), 'Cloud Security' (Secure Score, Regulatory compliance, Workload protections, Firewall Manager), and 'Management' (Environment settings, Security solutions, Workflow automation). The main content area has four cards: 'Secure score' (9 Azure subscriptions, 5 Assessed resources, 1 Active recommendations, 0 Security alerts), 'Unhealthy resources' (0, 'To harden these resources and improve your score, follow the security recommendations'), 'Current secure score' ('No data to display'), and 'Regulatory compliance' (empty). To the right, there's an 'Insights' section with 'Most prevalent recommendations (by resources)' (FTPS should be required in your web App, Web apps should request an SSL certificate for all incoming requests, Diagnostic logs should be enabled in App Service, Container registries should not allow unrestricted network access, all with count 2) and a 'Controls with the highest potential increase' section ('No data to display'). At the bottom, there's a 'Defender for Cloud community' section with a GitHub icon and a link to join the community on GitHub.

# Cryptocurrency mining and other-advanced attacks

Azure Defender for container registries (deprecated) can be enabled at the subscription level.

Once it's enabled:

- Microsoft Defender will then scan images that are pushed to the registry.
- Scan imported into the registry.
- Or any images pulled within the last 30 days.

There's a per-image charge for this feature.

When issues are found, a notification appears in the Microsoft Defender dashboard.

There are three triggers for image scans:

- On push (a new image is pushed to the registry)
- Recently pulled (any image pulled in the last 30 days)
- On import (when an image is imported from other locations like Docker Hub)

## Important

Microsoft Defender for container registries has been replaced with [Microsoft Defender for Containers](#). If you've already enabled Defender for container registries on a subscription, you can continue to use it. However, you won't get Defender for Containers' improvements and new features.

This plan is no longer available for subscriptions where it isn't already enabled.

To upgrade to Microsoft Defender for Containers, open the Defender plans page in the portal and enable the new plan:



Learn more about this change in [the release note](#).

# Purchasing Microsoft Defender

Microsoft Defender is available as part of the Microsoft *Enterprise Mobility + Security E5* offering and a standalone license.

You can acquire a license directly from the [Enterprise Mobility + Security pricing options](#) page or the Cloud Solution Provider (CSP) licensing model.

## Note

Microsoft Defender isn't available for purchase via the Azure portal. For more information about Microsoft Defender review the [Azure Defender | Microsoft Azure](#) webpage.

# OWASP and Dynamic Analyzers

## Plan Implement OWASP Secure Coding Practices

Completed 100 XP

- 1 minute

The starting point for secure development is to use secure-coding practices.

The [Open Web Application Security Project \(OWASP\)](#) is a global charitable organization focused on improving software security.

OWASP's stated mission is to make software security visible so that individuals and organizations can make informed decisions.

They offer impartial and practical advice.

OWASP regularly publishes a set of Secure Coding Practices. Their guidelines currently cover advice in the following areas:

- Input Validation
- Output Encoding
- Authentication and Password Management
- Session Management
- Access Control
- Cryptographic Practices
- Error Handling and Logging
- Data Protection
- Communication Security
- System Configuration
- Database Security
- File Management
- Memory Management
- General Coding Practices

OWASP also publishes an intentionally vulnerable web application called [The Juice Shop Tool Project](#) to learn about common vulnerabilities and see how they appear in applications.

It includes vulnerabilities from all the [OWASP Top 10](#).

In 2002, Microsoft underwent a company-wide re-education and review phase to produce secure application code.

The book, [Writing Secure Code by David LeBlanc, Michael Howard](#), was written by two people involved and provided detailed advice on writing secure code.

For more information, you can see the following:

- [The OWASP Foundation](#).
- [OWASP Secure Coding Practices Quick Reference Guide](#).
- [OWASP Code Review Guide](#).
- [OWASP Top 10](#).

## Explore OWASP ZAP penetration test

Completed 100 XP

- 2 minutes

ZAP is a free penetration testing tool for beginners to professionals. ZAP includes an API and a weekly docker container image to integrate into your deployment process.

Refer to the [OSWA ZAP VSTS Extension](#) repo for details on setting up the integration. Here we're going to explain the benefits of including it in your process.

The application CI/CD pipeline should run within a few minutes, so you don't want to include any long-running processes.

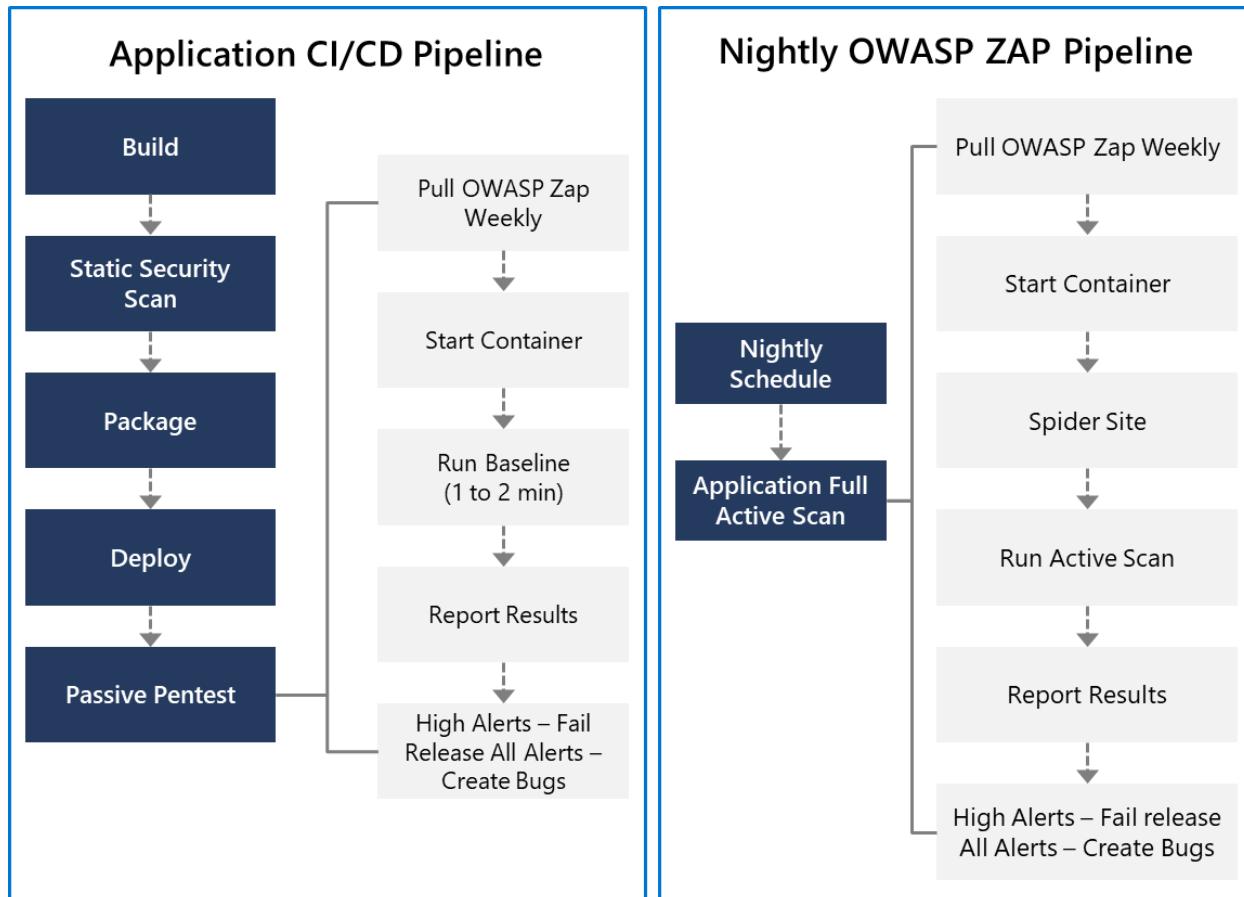
The baseline scan is designed to identify vulnerabilities within a couple of minutes, making it a good option for the application CI/CD pipeline.

The Nightly OWASP ZAP can spider the website and run the full-Active Scan to evaluate the most combinations of possible vulnerabilities.

OWASP ZAP can be installed on any machine in your network, but we like to use the OWASP Zap/Weekly docker container within Azure Container Services.

It allows for the latest updates to the image. It will enable the spin-up of multiple image instances so several applications within an enterprise can be scanned simultaneously.

The following figure outlines the steps for the Application CI/CD pipeline and the longer-running Nightly OWASP ZAP pipeline.



## Explore OWASP ZAP results and bugs

Completed 100 XP

- 2 minutes

Once the scans have completed, the Azure Pipelines release is updated with a report that includes the results and bugs are created in the team's backlog.

Resolved bugs will close if the vulnerability has been fixed and move back into in-progress if the vulnerability still exists.

The benefit of using this is that the vulnerabilities are created as bugs that provide actionable work that can be tracked and measured.

False positives can be suppressed using OWASP ZAP's context file, so only valid vulnerabilities are surfaced.

OWASP ZAP Nightly / Release-40

Summary Environments Artifacts Configuration General Commits Work items

Deploy Save Abandon

Total tests 6 Pass percentage 0% Run duration 0s

Expand all Collapse all Create bug

Test

- 0/6 Passed - OWASP ZAP Security Tests
- Incomplete or No Cache-control and Pragma HTTP Header Set
- Cookie No HttpOnly Flag
- Cookie Without Secure Flag
- Web Browser XSS Protection Not Enabled
- X-Content-Type-Options Header Missing
- X-Frame-Options Header Not Set

Even with continuous security validation running against every change to help ensure new vulnerabilities aren't introduced, hackers continuously change their approaches, and new vulnerabilities are being discovered.

Good monitoring tools allow you to help detect, prevent, and remediate issues discovered while your application runs in production.

Azure provides several tools that provide detection, prevention, and alerting using rules, such as [OWASP Top 10](#) and machine learning to detect anomalies and unusual behavior to help identify attackers.

Minimize security vulnerabilities by taking a holistic and layered approach to security, including secure infrastructure, application architecture, continuous validation, and monitoring.

DevSecOps practices enable your entire team to incorporate these security capabilities in the whole lifecycle of your application.

Establishing continuous security validation into your CI/CD pipeline can allow your application to stay secure while improving the deployment frequency to meet the needs of your business to stay ahead of the competition.

# Static analyzers

## Explore SonarCloud

Completed 100 XP

- 2 minutes

Technical debt can be classified as the measure between the codebase's current state and an optimal state.

Technical debt saps productivity by making code hard to understand, easy to break, and difficult to validate, creating unplanned work and ultimately blocking progress.

Technical debt is inevitable! It starts small and grows over time through rushed changes, lack of context, and discipline.

Organizations often find that more than 50% of their capacity is sapped by technical debt.

The hardest part of fixing technical debt is knowing where to start.

SonarQube is an open-source platform that is the de facto solution for understanding and managing technical debt.

We'll learn how to use SonarQube in a build pipeline to identify technical debt in this recipe.

## Getting ready

SonarQube is an open platform to manage code quality.

Originally famous in the Java community, SonarQube now supports over 20 programming languages.

The joint investments made by Microsoft and SonarSource make SonarQube easier to integrate with Pipelines and better at analyzing .NET-based applications.

You can read more about the capabilities offered by SonarQube here: <https://www.sonarqube.org/>.

SonarSource, the company behind SonarQube, offers a hosted SonarQube environment called as SonarCloud.

## Explore CodeQL in GitHub

Completed 100 XP

- 2 minutes

Developers use CodeQL to automate security checks.

CodeQL treats code like data that can be queried.

GitHub researchers and community researchers have contributed standard CodeQL queries, and you can write your own.

A CodeQL analysis consists of three phases:

- Creating a CodeQL database (based upon the code).
- Run CodeQL queries against the database.
- Interpret the results.

CodeQL is available as a command-line interpreter and an extension for Visual Studio Code.

For an overview of CodeQL, see [CodeQL Overview](#).

For the available tools, see [CodeQL Tools](#).

## Manage technical debt with SonarCloud and Azure DevOps

Completed 100 XP

- 60 minutes

**Estimated time:** 60 minutes.

**Lab files:** none.

## Scenario

In the context of Azure DevOps, the term **technical debt** represents suboptimal means of reaching tactical goals, which negatively affects the ability to achieve strategic objectives in software development and deployment. Technical debt affects productivity by making code hard to understand, prone to failures, time-consuming to change, and difficult to validate. Without proper oversight and management, technical debt can accumulate over time and significantly impact the overall quality of the software and the productivity of development teams in the longer term.

[SonarCloud](#) is a cloud-based code quality and security service. The main features of SonarCloud include:

- Support for 23 programming and scripting languages, including Java, JS, C#, C/C++, Objective-C, TypeScript, Python, ABAP, PLSQL, and T-SQL.
- There are thousands of rules to track down hard-to-find bugs and quality issues based on powerful static code analyzers.
- Cloud-based integrations with popular CI services, including Travis, Azure DevOps, BitBucket, and AppVeyor.
- Deep code analysis for exploring all source files in branches and pull requests, helping reach a green Quality Gate and promote the build.
- Speed and scalability.

In this lab, you'll learn how to integrate Azure DevOps with SonarCloud.

### Note

Before you run this lab, ensure that you can run Azure Pipelines. Due to the change to public projects that took place in February 2021, access to pipelines will need to be requested: <https://devblogs.microsoft.com/devops/change-in-azure-pipelines-grant-for-public-projects>

## Objectives

After completing this lab, you'll be able to:

- Set up an Azure DevOps project and CI build to integrate with SonarCloud.
- Analyze SonarCloud reports.
- Integrate static analysis into the Azure DevOps pull request process.

# Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).

# Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Configure Sonarcloud Setup.
- Exercise 2: Analyze SonarCloud reports.
- Exercise 3: Implement Azure DevOps pull request integration with SonarCloud.

# Software Composition Analysis

## Inspect and validate code bases for compliance

Completed 100 XP

- 3 minutes

Security for applications is critical. Every day, news services worldwide seem to carry stories about some company systems breached. More importantly, private company and customer data have been disclosed.

It has been happening for a long time. In many cases, it wasn't visible to the public. Private information was often disclosed, yet the people affected weren't even notified.

Governments worldwide frequently enact legislation to require information about breaches to become public and notifications to the affected.

So, what are the issues?

We need to protect information from being disclosed to people who shouldn't have access. But more importantly, we need to ensure that the data isn't altered or destroyed when it shouldn't be, and we need to make sure it's destroyed when it's supposed to be.

We need to make sure we properly authenticate who is accessing the data and that they have the correct permissions to do so. We need to find evidence when something has gone wrong through historical or archival data or logs.

There are many aspects to building and deploying secure applications.

- First, there's a general knowledge problem. Many developers and other staff assume they understand security, but they don't. Cybersecurity is a constantly evolving discipline. A program of ongoing education and training is essential.
- Second, we need to ensure that the code is created correctly and securely implements the required features, and we need to make sure that the features were designed with security in mind in the first place.

- Third, we need to ensure that the application follows the rules and regulations required to meet. We need to test it while building the code and retest it periodically, even after deployment.

It's commonly accepted that security isn't something you can add to an application or a system later.

Secure development must be part of the development life cycle. It's even more important for critical applications and those who process sensitive or highly confidential information.

Application security concepts haven't been a focus for developers in the past. Apart from the education and training issues, their organizations have emphasized the fast development of features.

However, with the introduction of DevOps practices, security testing is much easier to integrate. Rather than being a task done by security specialists, security testing should be part of the day-to-day delivery processes.

Overall, when the time for rework is taken into account, adding security to your DevOps practices can reduce the overall time to develop quality software.

## Explore software composition analysis (SCA)

Completed 100 XP

- 4 minutes

Two crucial areas of the Secure DevOps pipeline are Package management and Open-Source Software OSS components.

### Package management

Just as teams use version control as a single source of truth for source code, Secure DevOps relies on a package manager as the unique source of binary components.

Using binary package management, a development team can create a local cache of approved components and a trusted feed for the Continuous Integration (CI) pipeline.

In Azure DevOps, **Azure Artifacts** is an integral part of the component workflow for organizing and sharing access to your packages. Azure Artifacts allows you to:

- Keep your artifacts organized. Share code easily by storing Apache Maven, npm, and NuGet packages together. You can store packages using Universal Packages, eliminating keeping binaries in Git.
- Protect your packages. Keep every public source package you use (including packages from npmjs and NuGet .org) safe in your feed where only you can delete it and it's backed by the enterprise-grade Azure Service Level Agreement (SLA).
- Integrate seamless package handling into your Continuous Integration (CI)/Continuous Development (CD) pipeline. Easily access all your artifacts in builds and releases. Azure Artifacts integrates natively with the Azure Pipelines CI/CD tool.

For more information about Azure Artifacts, visit the webpage. [What are Azure Artifacts?](#)

## Versions and compatibility

The following table lists the package types supported by Azure Artifacts. The availability of each package in *Azure DevOps Services* is also displayed.

The following table details the compatibility of each package with specific versions of the Azure DevOps Server, previously known as *Team Foundation Server* (TFS).

Expand table

| Feature   | Azure DevOps Services | TFS                           |
|-----------|-----------------------|-------------------------------|
| NuGet     | Yes                   | TFS 2017                      |
| npm       | Yes                   | TFS 2017 update one and later |
| Maven     | Yes                   | TFS 2017 update one and later |
| Gradle    | Yes                   | TFS 2018                      |
| Universal | Yes                   | No                            |
| Python    | Yes                   | No                            |

Maven, npm, and NuGet packages can be supported with teams of any size from public and private sources. Azure Artifact comes with Azure DevOps, but the extension is also available from the Visual Studio Marketplace.



P

## Parts-Unlimited



Overview



Boards



Repos



Pipelines



Test Plans



Artifacts

## Connect to feed

## NuGet

npm

Maven

Gradle

Universal

Python

**Note**

After publishing a particular package version to a feed, that version number is permanently reserved.

**Note**

You can't upload a newer revision package with that same version number or delete that version and upload a new package with the same version number. The published version is immutable.

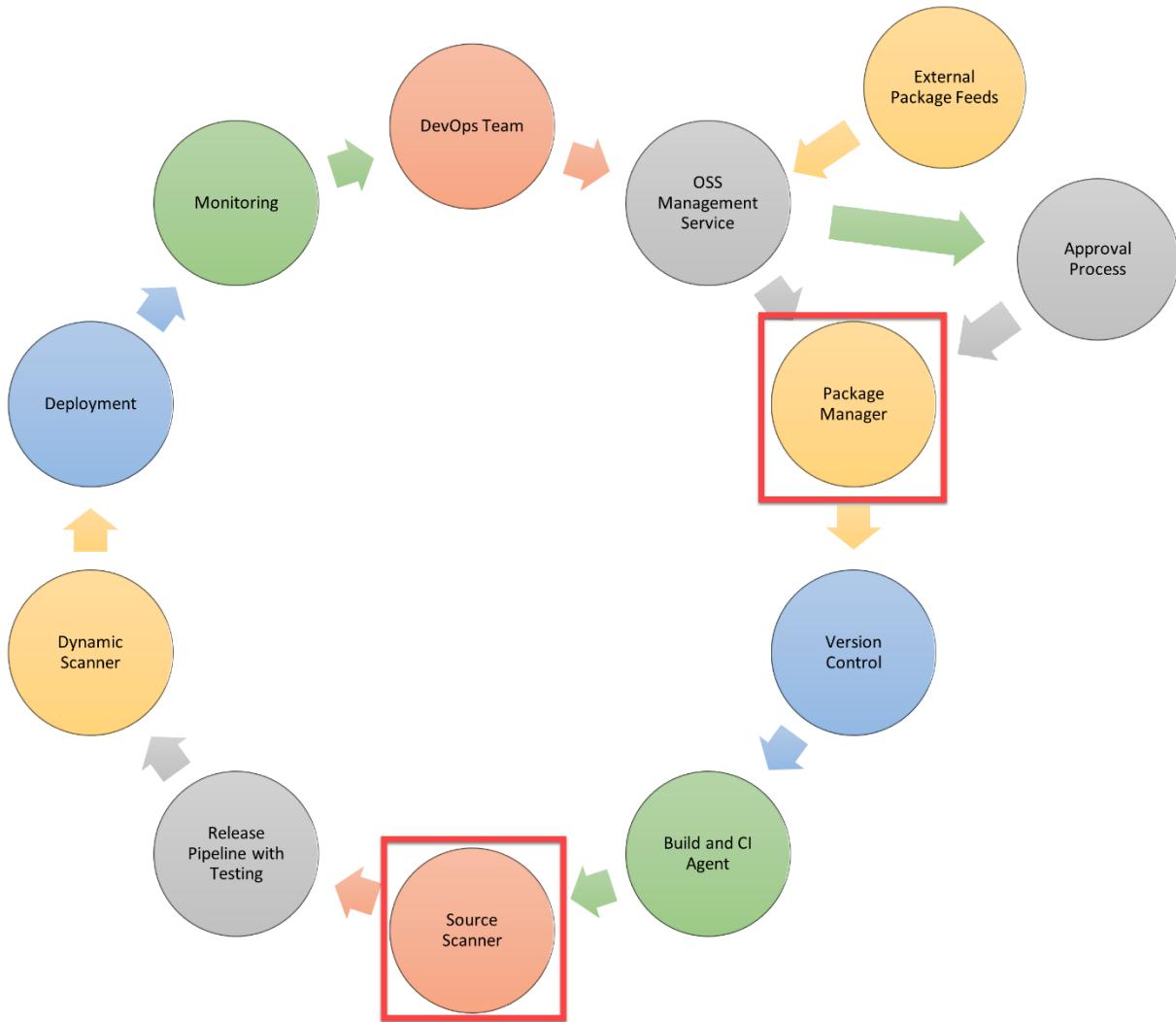
## The Role of OSS components

Development work is more productive because of the wide availability of reusable Open-source software (OSS) components.

This practical approach to reuse includes runtimes, which are available on Windows and Linux operating systems such as Microsoft .NET Core and Node.js.

However, OSS component reuse comes with the risk that reused dependencies can have security vulnerabilities. As a result, many users find security vulnerabilities in their applications because of the Node.js package versions they consume.

OSS offers a new-concept-called *Software Composition Analysis (SCA)* to address these security concerns, shown in the following image.



When consuming an OSS component, whether you're creating or consuming dependencies, you'll typically want to follow these high-level steps:

1. Start with the latest, correct version to avoid old vulnerabilities or license misuse.
2. Validate that the OSS components are the correct binaries for your version. In the release pipeline, validate binaries to ensure accuracy and keep a traceable bill of materials.
3. Get notifications of component vulnerabilities immediately, correct them, and redeploy the component automatically to resolve security vulnerabilities or license misuse from reused software.

## Integrate Mend with Azure Pipelines

Completed 100 XP

- 4 minutes

Visual Studio Code Marketplace is an important site for addressing Secure DevOps issues. You can integrate specialist security products into your Azure DevOps pipeline.

Having a full suite of extensions that allow a seamless integration into Azure Pipelines is invaluable.

## Mend

The Mend extension is available on the Azure DevOps Marketplace. Using Mend, you can integrate extensions with your CI/CD pipeline to address Secure DevOps security-related issues.

The Mend extension specifically addresses open-source security, quality, and license compliance concerns for a team consuming external packages.

Because most breaches target known vulnerabilities in standard components, robust tools are essential to securing complex open-source components.

## Continuously detect all open-source components in your software

Mend automatically detect all open-source components—including their transitive dependencies—every time you run a build.

It means you can generate a comprehensive inventory report within minutes based on the last build you ran.

It also gives complete visibility to your security, DevOps, and legal teams into your organization's software development process.

Home > Reports > Inventory

**Inventory Report** All Products All Projects **Apply** Actions Export

Filter Library Name Value **Filter**

| Library Name                  | Type               | Description   | Licenses                               | Match Type     | Occurrences       | Actions |
|-------------------------------|--------------------|---|--|----------------|-------------------|---------|
| json-schema-0.2.3.tgz         | javascript/Node.js | JSON Schema validation and specifications   | Academic 2.1, BSD 3                    | Filename match | 1 project details |         |
| avalon-framework-4.1.3.jar    | Java               |   | Apache 1.1                             | Exact match    | 1 project details |         |
| oro-2.0.8.jar                 | Java               |   | Apache 1.1                             | Exact match    | 1 project details |         |
| commons-el-5.5.23.jar         | Java               | An implementation of the Expression Language (EL).  | Apache 1.1                             | Exact match    | 1 project details |         |
| xpp3-1.1.4c.jar               | Java               | MX4P1 is a stable XMLPull parsing engine that is based on ideas from XPP and in particular XPP2 but completely revised and rewritten to take the best advantage of latest JIT JVMs such as Hotspot in JDK 1.4+. | Apache 1.1, Indiana University Extr... | Exact match    | 1 project details |         |
| commons-beanutils-1.8.0.jar   | Java               | BeanUtils provides an easy-to-use but flexible wrapper around reflection and introspection.   | Apache 2.0                             | Exact match    | 1 project details |         |
| commons-logging-1.1.1.jar     | Java               | Commons Logging is a thin adapter allowing configurable bridging to other, well known logging systems.  | Apache 2.0                             | Exact match    | 1 project details |         |
| commons-collections-3.2.1.jar | Java               | Collection utility classes.   | Apache 2.0                             | Exact match    | 1 project details |         |
| commons-digester-1.7.jar      | Java               | The Digester package lets you configure an XML->Java object mapping module which triggers certain actions called rules.   | Apache 2.0                             | Exact match    | 1 project details |         |

## Receive alerts on open-source security vulnerabilities

Mend automatically generates an alert and provides targeted remediation guidance when a new security vulnerability is discovered.

It can include links to patches, fixes, relevant source files, even recommendations to change system configuration to prevent exploitation.

Home > Reports > Vulnerabilities

**Vulnerabilities** High: 151 Medium: 103 Low: 8 All Products All Projects **Apply** Export Excel XML JSON

Filter Severity Value **Filter**

| Severity | Library              | Occurrences       | Vulnerability Id | CVSS 3 Score | CVSS 2 Sc... | Published  | Modified   | Top Fix   |
|----------|----------------------|-------------------|------------------|--------------|--------------|------------|------------|---|
| High     | xstream-1.4.10.jar   | 1 project details | CVE-2021-21345   | 9.9          | 6.5          | 23-03-2021 | 06-04-2021 | Upgrade to version com.thoughtworks.xstream:xstream:1.4.16<br>1 more fix available  |
| High     | bcprov-jdk14-136.jar | 1 project details | CVE-2018-5382    | 9.8          | 7.5          | 16-04-2018 | 02-04-2021 | Upgrade to version org.bouncycastle:bcprov-ext-jdk14:1.47<br>org.bouncycastle:bcprov-ext-jdk15on:1.47<br>2 more fixes available |
| High     | xstream-1.4.10.jar   | 1 project details | CVE-2013-7285    | 9.8          | 7.5          | 15-05-2019 | 21-10-2020 | Upgrade to version 1.4.7,1.4.11<br>4 more fixes available   |
| High     | xstream-1.4.10.jar   | 1 project details | CVE-2019-10173   | 9.8          | 7.5          | 23-07-2019 | 16-03-2021 | Upgrade to version 1.4.11<br>2 more fixes available   |
| High     | xstream-1.4.10.jar   | 1 project details | CVE-2021-21344   | 9.8          | 7.5          | 23-03-2021 | 06-04-2021 | Upgrade to version com.thoughtworks.xstream:xstream:1.4.16<br>1 more fix available  |
| High     | xstream-1.4.10.jar   | 1 project details | CVE-2021-21346   | 9.8          | 7.5          | 23-03-2021 | 06-04-2021 | Upgrade to version com.thoughtworks.xstream:xstream:1.4.16<br>1 more fix available  |
| High     | xstream-1.4.10.jar   | 1 project details | CVE-2021-21347   | 9.8          | 7.5          | 23-03-2021 | 06-04-2021 | Upgrade to version com.thoughtworks.xstream:xstream:1.4.16<br>1 more fix available  |
| High     | xstream-1.4.10.jar   | 1 project details | CVE-2021-21350   | 9.8          | 7.5          | 23-03-2021 | 06-04-2021 | Upgrade to version com.thoughtworks.xstream:xstream:1.4.16<br>1 more fix available  |
| High     | Pillow-5.2.0.tar.gz  | 1 project details | CVE-2020-5311    | 9.8          | 7.5          | 03-01-2020 | 10-07-2020 | Upgrade to version Pillow - 6.2.2<br>2 more fixes available   |
| High     | Pillow-5.2.0.tar.gz  | 1 project details | CVE-2020-5312    | 9.8          | 7.5          | 03-01-2020 | 10-07-2020 | Upgrade to version Pillow - 6.2.2<br>2 more fixes available   |

## Automatically enforce open-source security and license compliance policies

According to a company's policies, Mend automatically approves, rejects, or triggers a manual approval process every time a new open-source component is added to a build.

Developers can set up policies based on parameters such as security-vulnerability severity, license type, or library age.

When a developer adds a problematic open-source component, the service will alert and fail the build.

For searching online repositories such as GitHub and Maven Central, Mend also offers an innovative browser extension.

Before choosing a new component, a developer can review its security vulnerabilities, quality, license issues, and whether it fits their company's policies.

## Implement GitHub Dependabot alerts and security updates

Completed 100 XP

- 2 minutes

### Alerts

GitHub Dependabot detects vulnerable dependencies and sends Dependabot alerts about them in several situations:

- A new vulnerability is added to the GitHub Advisory database.
- New vulnerability data from Mend is processed.
- Dependency graph for a repository changes.

Alerts are detected in public repositories by default but can be enabled for other repositories.

Notifications can be sent via standard GitHub notification mechanisms.

For more information on Dependabot Alerts, see [About alerts for vulnerable dependencies](#).

See Supported package ecosystems for details on the provided packages that alerts can be generated.

For notification details, see: [Configuring notifications](#).

## Security updates

A key advantage of Dependabot security updates is that they can automatically create pull requests.

A developer can then review the suggested update and triage what is required to incorporate it.

For more information on automatic security updates, see [About GitHub Dependabot security updates](#).

## Integrate software composition analysis checks into pipelines

100 XP

- 4 minutes

Security scanning used to be thought of as an activity that was completed once per release by a dedicated security team whose members had little involvement with other groups.

This practice creates a dangerous pattern in which security specialists find large batches of issues at the exact time when developers are under the most pressure to release a software product.

The pressure often results in software deployment with security vulnerabilities that need to be addressed after a product has been released, integrating scanning into a team's workflow at multiple points along the development path. Secure DevOps can help make all quality-assurance activities, including security, continuous and automated.

## Pull request code scan analysis integration.

DevOps teams can submit proposed changes to an application's (main) codebase using pull requests (PRs). To avoid introducing new issues, developers need to verify the effects of the code changes before creating a PR. A PR is typically made for each small change in a DevOps process. Changes are continuously merged with the main codebase to keep the main codebase up to date. Ideally, a developer should check for security issues before creating a PR.

Azure Marketplace extensions that help integrate scans during PRs include:

- [Mend](#). Helps validate dependencies with its binary fingerprinting.
- [Checkmarx](#). Provides an incremental scan of changes.
- [Veracode](#). Implements the concept of a developer sandbox.
- [Black Duck by Synopsis](#). An auditing tool for open-source code to help identify, fix, and manage compliance.

These extensions allow developers to experiment with changes before submitting them as a PR.

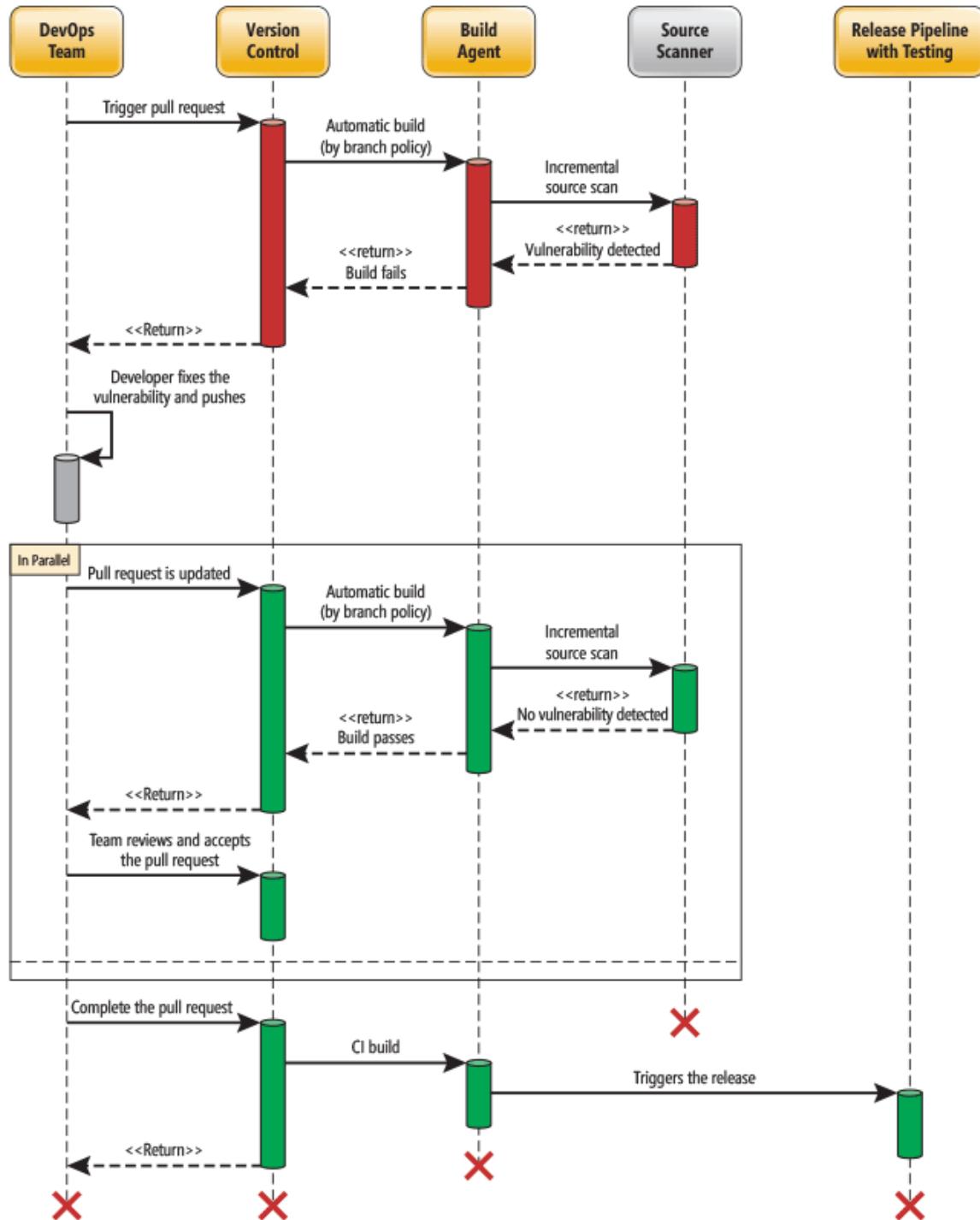
## Build and release definition code scan, analysis, and integration

Developers need to optimize CI for speed to get immediate feedback about build issues. Code scanning can be performed quickly enough to integrate the CI build definition, preventing a broken build. It enables developers to restore a build's status to ready/green by fixing potential issues immediately.

At the same time, the CD needs to be thorough. In Azure DevOps, the CD is typically managed through release definitions (which progress the build output across environments) or other build definitions.

Build definitions can be scheduled (daily) or triggered with each commit. In either case, the build definition can do a longer static analysis scan (as illustrated in the following image).

You can scan the complete code project and review any errors or warnings offline without blocking the CI flow.



## Examine tools for assess package security and license rate

100 XP

- 2 minutes

Several tools are available from third parties to help assess software packages' security and license rating.

As discussed in the previous section, one approach by these tools is to provide a centralized artifact repository.

Scanning can be done at any time, inspecting the packages part of the repository.

The second approach uses tooling that scans the packages used in a build pipeline.

During the build process, the tool can scan the packages by the build, giving instantaneous feedback on the packages in use.

## Inspect packages in the delivery pipeline

While running a delivery pipeline, there's tooling available to do security scans on packages, components, and source code. Often such tooling will use the build artifacts during the build process and do scans. The tooling can either work on a local artifact repository or the intermediary build output. Some examples for each are products like:

Expand table

| Tool        | Type                        |
|-------------|-----------------------------|
| Artifactory | Artifact repository         |
| SonarQube   | A static code analysis tool |
| Mend (Bolt) | Build scanning.             |

## Configure pipeline

The configuration of the scanning for license types and security vulnerability in the pipeline is done by using appropriate build tasks in your DevOps tooling. For Azure DevOps, these are build pipeline tasks.

## Interpret alerts from scanner tools

100 XP

- 2 minutes

To correctly interpret the results of scanning tools, you need to be aware of some aspects:

- **False positives** It's essential to verify the findings to be real positives in the scan results. The tooling is an automated way to scan and might be misinterpreting specific vulnerabilities. In the triaging of the finding in the scan results, you should be aware that some findings might not be correct. Such results are called false positives, established by human interpretation and expertise. One must not declare a result a false positive too quickly. On the other hand, scan results aren't guaranteed to be 100% accurate.
- **Security bug bar** Most likely, many security vulnerabilities will be detected—some of these false positives, but still many findings. More findings can often be handled or mitigated, given a certain amount of time and money. In such cases, there must be a security bug bar indicating the level of vulnerabilities that must be fixed before the security risks are acceptable enough to take the software into production. The bug bar makes sure that it's clear what must be taken care of and what might be done if time and resources are left.

The results of the tooling scan will be the basis for selecting what work remains to be done before the software is considered stable and done.

By setting a security bug bar in the Definition of Done and specifying the allowed license ratings, one can use the reports from the scans to find the work for the development team.

## Implement security and compliance in an Azure Pipeline

100 XP

- 45 minutes

**Estimated time:** 45 minutes.

**Lab files:** none.

## Scenario

In this lab, you'll use **Mend Bolt (formerly WhiteSource) with Azure DevOps** to automatically detect vulnerable open source components, outdated libraries, and license compliance issues in your code. You'll use WebGoat, an intentionally insecure web application maintained by OWASP designed to illustrate common web application security issues.

[Mend](#) is the leader in continuous open source software security and compliance management. Mend integrates into your build process, irrespective of your programming languages, build tools, or development environments. It works automatically, continuously, and silently in the background, checking your open source components' security, licensing, and quality against Mend constantly updated definitive database of open source repositories.

Mend provides Mend Bolt, a lightweight open source security and management solution developed specifically for integrating Azure DevOps.

### Note

Mend Bolt works per project and doesn't offer real-time alert capabilities, which requires a **Full platform**.

Mend Bolt is recommended for larger development teams that want to automate their open source management throughout the entire software development lifecycle (from the repositories to post-deployment stages) and across all projects and products.

Azure DevOps integration with Mend Bolt will enable you to:

- Detect and remedy vulnerable open source components.
- Generate comprehensive open source inventory reports per project or build.
- Enforce open source license compliance, including dependencies' licenses.
- Identify outdated open source libraries with recommendations to update.

## Objectives

After completing this lab, you'll be able to:

- Activate Mend Bolt.
- Run a build pipeline and review the Mend security and compliance report.

# Requirements

- This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](#).
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](#).

# Exercises

During this lab, you'll complete the following exercises:

- Exercise 0: Configure the lab prerequisites.
- Exercise 1: Implement Security and Compliance in an Azure DevOps pipeline using Mend Bolt.

# Implement open-source software

## Explore how software is built

Completed 100 XP

- 1 minute

Let us look at using open-source software in building software.

### Using open-source software

Packages contain components that are built from source code. Open-source code is publicly available for inspection, reuse, and contribution.

Most commonly, open-source projects indicate how the sources can be used and distributed afterward. A license agreement comes with the source code and specifies what can and cannot be done.

Software today is built by using components. These components are created in part by the team that is writing the entire software solution.

Some dependencies are on components created and made available by other teams, third-party companies, and the community. The packages that contain the components are a formalized way for distribution.

On average, the built software solution is around 80% based on existing components and maintained outside of the project.

The rest of the solution consists of your code with business logic and specifics for the functional requirements. Plus, "glue" code that binds the components and your code. The components can be a commercial offering or free of charge.

A considerable part of the publically available and free components are community efforts to offer reusable components for everyone to use and build software. The persons creating and maintaining these components often also make the source code available.

It's open-source code as opposed to closed source. A closed source means that the source code isn't available, even though components are available.

# What is open-source software

Completed 100 XP

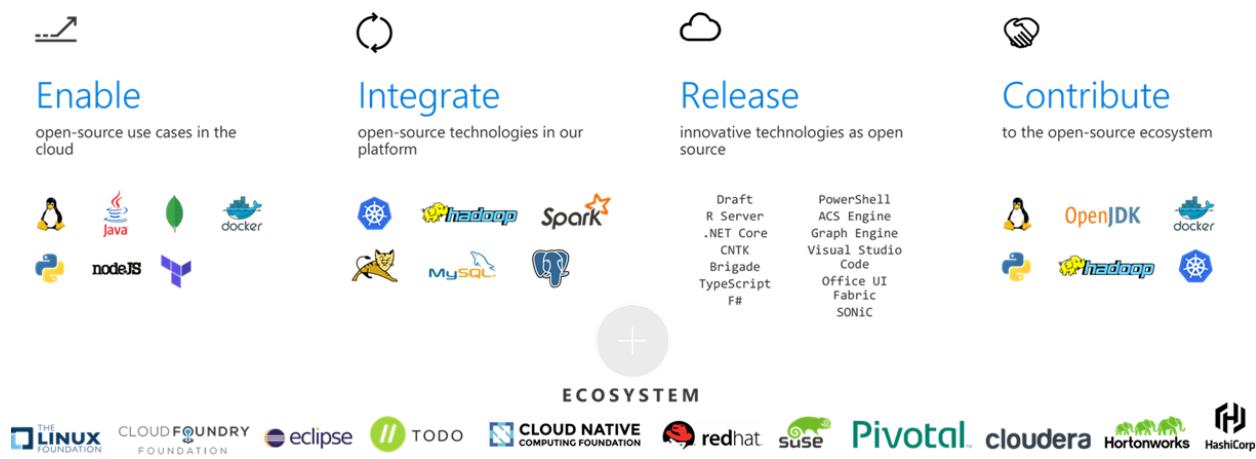
- 1 minute

Wikipedia defines open-source software as follows:

*"Open-source software is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose."*

The related open-source software development is a collaborative form of software development involving multiple contributors. Together they create and maintain software and source code using open sources. The use of open-source software is widely adopted now.

Microsoft itself has also-embraced open-source software in its software and the development platforms they offer.



The .NET platforms, such as the original .NET Framework and even more so .NET Core, use several components created by the open-source community and not Microsoft itself. In [ASP.NET](#) and [ASP.NET](#) Core, many of the frontend development libraries are open-source components, such as jQuery, Angular, and React.

Instead of creating new components themselves, the teams at Microsoft are using the open-source components and taking a dependency on them.

The teams also contribute and invest in the open-source components and projects, joining in on the collaborative effort. Besides adopting external open-source software, Microsoft has also made significant parts of its software available as open-source.

.NET is a perfect example of how Microsoft has changed its posture towards open source. It has made the codebase for the .NET Framework and .NET Core available and many other components.

The .NET Foundation aims to advocate the needs and evangelize the benefits of the .NET platform. And promote the use of .NET open source for developers.

For more information, see the [.NET Foundation website](#).

## Explore corporate concerns with open-source software components

Completed 100 XP

- 2 minutes

In summary, modern software development, including the Microsoft developer platform and ecosystem, implies open-source components.

It has implications for companies that build software, either commercially or for internal use.

The inclusion of software components that are not built by the companies themselves means no complete control over the sources.

Being responsible for the source code used in components used within a company means that you must accept its risks. The concerns are that source code component:

- **Are of low quality.** It would impact the maintainability, reliability, and performance of the overall solution.
- **Have no active maintenance.** The code wouldn't evolve or be alterable without copying the source code, effectively forking away from the origin.
- **Contain malicious code.** The entire system that includes and uses the code will be compromised. Potentially the whole company's IT and infrastructure is affected.

- **Have security vulnerabilities.** The security of a software system is as good as its weakest part. Using source code with vulnerabilities makes the entire system susceptible to attack by hackers and misuse.
- **Have unfavorable licensing restrictions.** The effect of a license can affect the entire solution that uses the open-source software.

The companies will have to make a trade-off: its developers want to use open-source software components, allowing them to speed up development and use modern frameworks, libraries, and practices.

On the other hand, giving the developers and projects the freedom to include open-source software should not put the company at risk.

The company's challenges are finding a way to keep the developers empowered and free to choose technology to use while making sure the risks for the company are managed as well as possible.

Other challenges come from companies that offer open-source software to the public.

These challenges include having a business model around the open-source, when to publish open-source code and how to deal with community contributions.

The fact that your source code is open doesn't imply that anyone can make changes to it.

There can be contributions from community collaboration, but a company doesn't necessarily have to accept it. It's referred to as-closed open-source.

Suggestions for change are welcome, but the maintainers are the ones that carry out the actual changes.

## Introduction to open-source licenses

Completed 100 XP

- 1 minute

A license agreement accompanies open-source software and the related source code.

The license describes how the source code and the components built from it can be used and how any software created with it should deal with it.

According to the open-source definition of [OpenSource.org](#), a license shouldn't:

- Discriminate against persons or groups.
- Discriminate against fields of endeavor.
- Be specific to a product.
- Restrict other software.
- And more - See the [Open Source Definition](#).

To cover the exact terms of a license, several types exist. Each type has its specifics and implications, which we'll cover in the next part.

Even though multiple contributors generally develop open-source software from the community, it doesn't guarantee that it's secure and without vulnerabilities.

Multiple reviewers discover chances, but the discovery might not be immediate or before being consumed by others.

Since the source code is open-source, people with malicious intent can also inspect the code for vulnerabilities and exploit it when possible.

In that regard, it's both a blessing and a curse that open-source software has source code available for others.

## Explore common open-source licenses

Completed 100 XP

- 1 minute

In the current and previous units, we've talked about software components from the perspective of packages.

Packages are the formalized ways to distribute software components.

The licensing types and concerns about vulnerabilities extend to the packages, as these contain the components.

## Types of licenses

There are multiple licenses used in open-source, and they're different.

The license spectrum is a chart showing licenses from the developer's perspective and the implications of use for downstream requirements imposed on the overall solution and source code.



On the left side, there are the "attribution" licenses. They're permissive and allow practically every type of use by the software that consumes it. An example is building commercially available software, including the components or source code under this license.

The only restriction is that the original attribution to the authors remains included in the source code or as part of the downstream use of the new software. The right side of the spectrum shows the "copyleft" licenses.

These licenses are considered viral, as the use of the source code and its components, and distribution of the complete software, implies that all source code using it should follow the same license form.

The viral nature is that the use of the software covered under this license type forces you to forward the same license for all work with or on the original software.

The middle of the spectrum shows the "downstream" or "weak copyleft" licenses. It also requires that it must do so under the same license terms when the covered code is distributed.

Unlike the copyleft licenses, it doesn't extend to improvements or additions to the covered code.

# Examine license implications and ratings

Completed 100 XP

- 1 minute

Any form of software that uses code or components must consider the license type that is covered.

For companies, it becomes essential to know the types of licenses for components and packages that developers choose to use.

When these include even one viral license, it requires that all software uses the same license.

Any intellectual property you might have must be made public and open source according to the terms of the viral license type.

It has a tremendous impact on the project's source code and the company that produces it.

## License rating

Licenses can be rated by the impact that they have. When a package has a specific type of license, the use of the package implies keeping to the requirements of the package.

The license's impact on the downstream use of the code, components, and packages can be rated as High, Medium, and Low, depending on the copy-left, downstream, or attribution nature of the license type.

For compliance reasons, a high license rating can be considered a risk for compliance, intellectual property, and exclusive rights.

## Package security

The use of components creates a software supply chain.

The resultant product is a composition of all its parts and components.

It applies to the security level of the solution as well. So, like license types, it's essential to know how secure the components being used are.

If one of the components used isn't secure, then the entire solution isn't either.