

AZ-400T02

Implementing Continuous Integration



Contents

■	Module 0 Welcome	1
	Start Here	1
■	Module 1 Implementing Continuous Integration in an Azure DevOps Pipeline	7
	Continuous Integration Overview	7
	Implementing a Build Strategy	13
	Module 1 Review Questions	18
■	Module 2 Managing Code Quality and Security Policies	21
	Managing Code Quality	21
	Managing Security Policies	26
	Module 2 Review Questions	29
■	Module 3 Implementing a Container Strategy	31
	Implementing a Container Build Strategy	31
	Module 3 Review Questions	39



Module 0 Welcome

Start Here

Azure DevOps Curriculum

Welcome to the **Implementing Continuous Integration** course. This course is part of a series of courses to help you prepare for the AZ-400, **Microsoft Azure DevOps Solutions**¹ certification exam.

The DevOps certification exam is for DevOps professionals who combine people, process, and technologies to continuously deliver valuable products and services that meet end user needs and business objectives. DevOps professionals streamline delivery by optimizing practices, improving communications and collaboration, and creating automation. They design and implement strategies for application code and infrastructure that allow for continuous integration, continuous testing, continuous delivery, and continuous monitoring and feedback.

Exam candidates must be proficient with Agile practices. They must be familiar with both Azure administration and Azure development and experts in at least one of these areas. Azure DevOps professionals must be able to design and implement DevOps practices for version control, compliance, infrastructure as code, configuration management, build, release, and testing by using Azure technologies.

AZ-400 Study Areas	Weights
Implement DevOps Development Processes	20-25%
Implement Continuous Integration	10-15%
Implement Continuous Delivery	10-15%
Implement Dependency Management	5 -10%
Implement Application Infrastructure	15-20%
Implement Continuous Feedback	10-15%
Design a DevOps Strategy	20-25%

There are seven exam study areas. Each study area has a corresponding course. While it is not required that you have completed any of the other courses in the DevOps series before taking this course, it is

¹ <https://www.microsoft.com/en-us/learning/exam-AZ-400.aspx>

highly recommended that you start with the first course in the series, and progress through the courses in order.

✓ This course will focus on preparing you for the **Implement Continuous Integration** area of the AZ-400 certification exam.

About this Course

Course Description

This course provides knowledge and skills to implement the DevOps practices of continuous integration. Students will learn how to implement continuous integration in an Azure DevOps pipeline, how to manage code quality and security principles, and how to implement a container build strategy.

Level: Intermediate

Audience

Students in this course are interested in DevOps continuous integration processes or in passing the Microsoft Azure DevOps Solutions certification exam.

Prerequisites

- Students should have fundamental knowledge about Azure, version control, Agile software development, and core software development principles. It would be helpful to have experience in an organization that delivers software.
- It is recommended that you have experience working in an IDE, as well as some knowledge of the Azure portal. However, students who may not have a technical background in these technologies, but who are curious about DevOps practices as a culture shift, should be able to follow the procedural and expository explanations of continuous integration regardless.

Expected learning objectives

- Explain why continuous integration matters
- Implement continuous integration using Azure DevOps
- Configure builds and the options available
- Create an automated build workflow
- Integrate other build tooling with Azure DevOps
- Create hybrid build processes
- Describe what is meant by code quality and how it is measured
- Detect code smells
- Integrate automated tests for code quality
- Report on code coverage during testing
- Add tooling to measure technical debt
- Detect open source and other licensing issues
- Implement a container build strategy

Syllabus

This course includes content that will help you prepare for the Microsoft Azure DevOps Solution certification exam. Other content is included to ensure you have a complete picture of DevOps. The course content includes a mix of videos, graphics, reference links, module review questions, and hands-on labs.

Module 1 – Implementing Continuous Integration in an Azure DevOps Pipeline

In this module, you'll be introduced to continuous integration principles including: benefits, challenges, build best practices, and implementation steps. You will also learn about implementing a build strategy with workflows, triggers, agents, and tools. Content includes:

- Lesson: Continuous Integration Overview
- Lab: Enabling Continuous Integration with Azure Pipelines
- Lesson: Implementing a Build Strategy
- Lab: Creating a Jenkins Build Job and Triggering Continuous Integration

Module 2 – Managing Code Quality and Security Policies

In this module, you will learn how to manage code quality including: technical debt, SonarCloud, and other tooling solutions. You will also learn how to manage security policies with open source, OWASP, and WhiteSource Bolt. Content includes:

- Lesson: Managing Code Quality
- Lab: Managing Technical Debt with Azure DevOps and SonarCloud
- Lesson: Managing Security Policies
- Lab: Checking Vulnerabilities using WhiteSource Bolt and Azure DevOps

Module 3 – Implementing a Container Build Strategy

In this module, you will learn how to implement a container strategy including how containers are different from virtual machines and how microservices use containers. You will also learn how to implement containers using Docker. Content includes:

- Lesson: Implementing a Container Build Strategy
- Lab: Existing .NET Applications with Azure and Docker Images

✓ This course uses the **Microsoft DevOps Lab Environment**² to provide a hands-on learning environment.

Lab Environment Setup

We highly recommend that you complete the assigned hands-on lab work. To do the hands-on labs, you will need to complete the following steps.

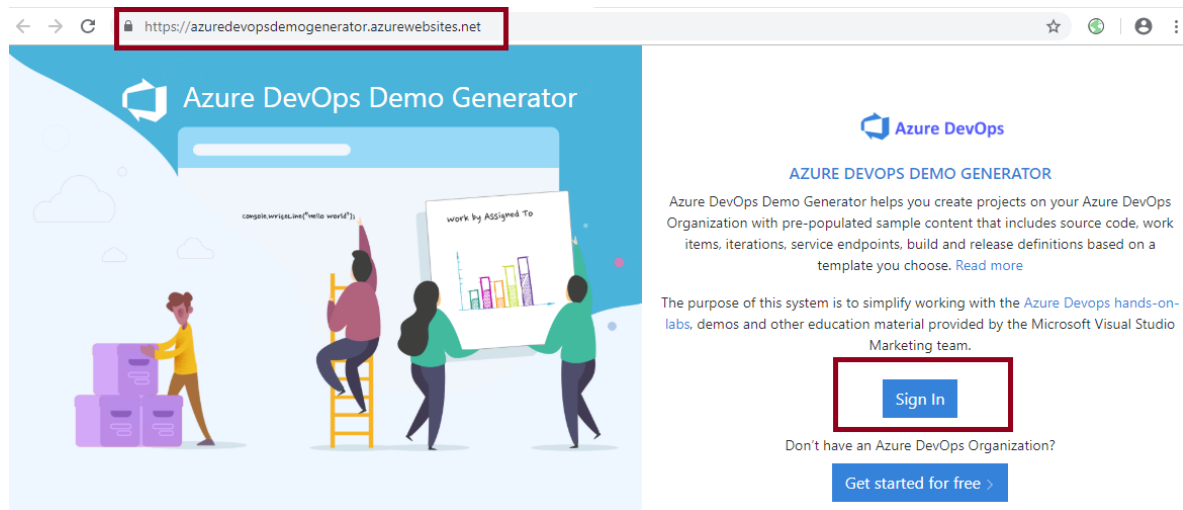
1. Sign up for a free **Azure DevOps account**³. Use the Sign up for a free account button to create your account. If you already have an account proceed to the next step.
2. Sign up for free **Azure Account**⁴. If you already have an account proceed to the next step.

² <https://azuredevopslabs.com/>

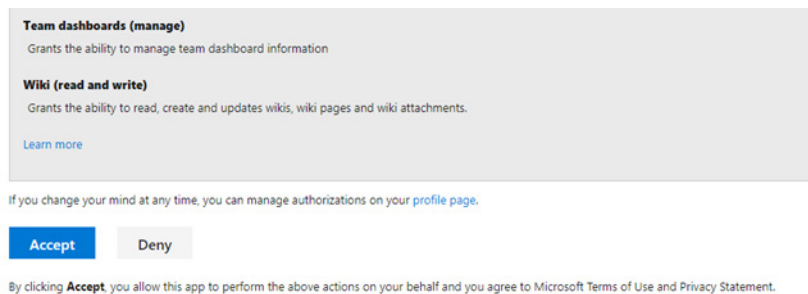
³ <https://www.azuredevopslabs.com/>

⁴ <https://azure.microsoft.com/en-us/free/>

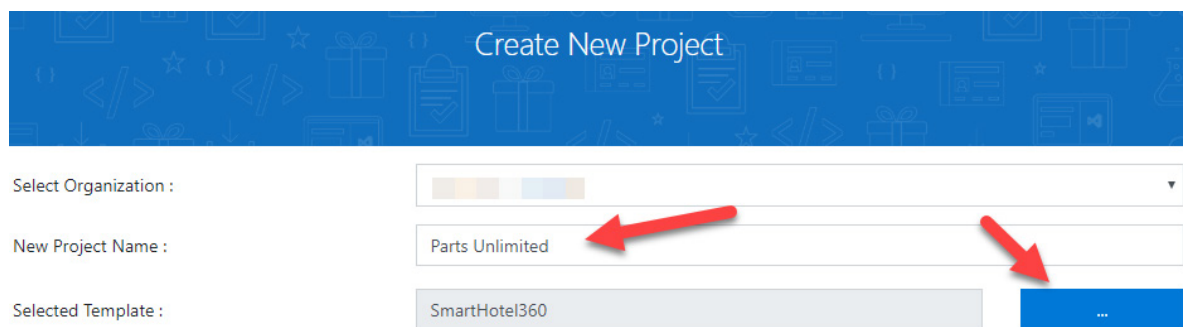
3. To make it easier to get set up for testing Azure DevOps, a **Azure DevOps Generator Demo**⁵ program has been created. Click the **Sign In** button and sign in with your Azure DevOps account.



1. You will then be asked to confirm that the generator site can have permission to create objects in your Azure DevOps account.

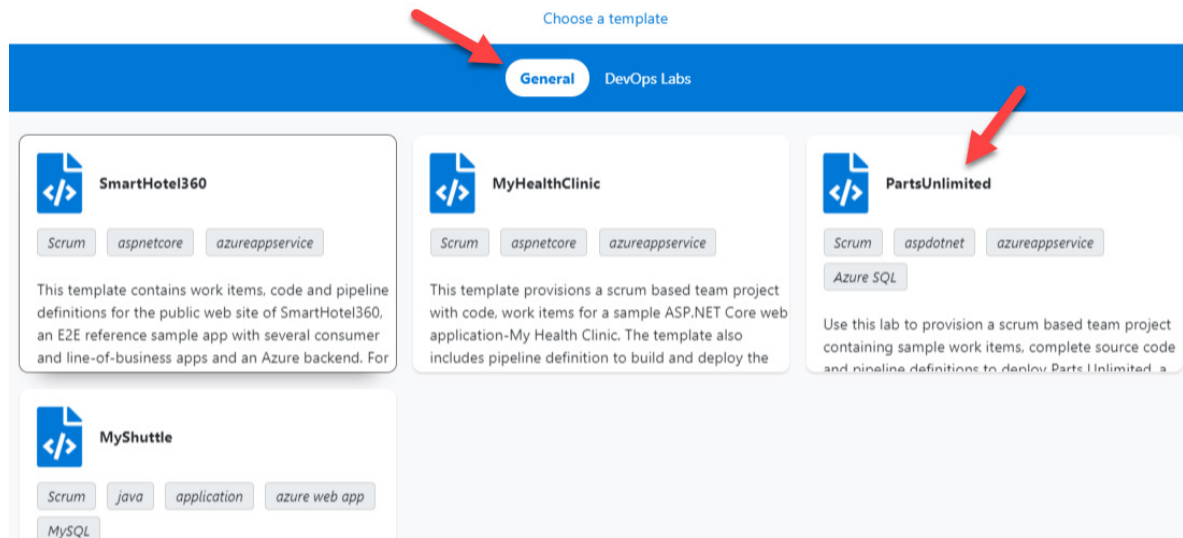


1. If you agree, click the **Accept** button and you should be greeted by the Create New Project screen:



1. Select the appropriate organization (if you have more than one) and enter **Parts Unlimited** as the **New Project Name**, then click the ellipsis to view the available templates. These will change over time but you should see a screen similar to the following:

⁵ <https://azuredevopsdemogenerator.azurewebsites.net/>



1. From the **General** tab, choose **PartsUnlimited**, then click **Select Template**.

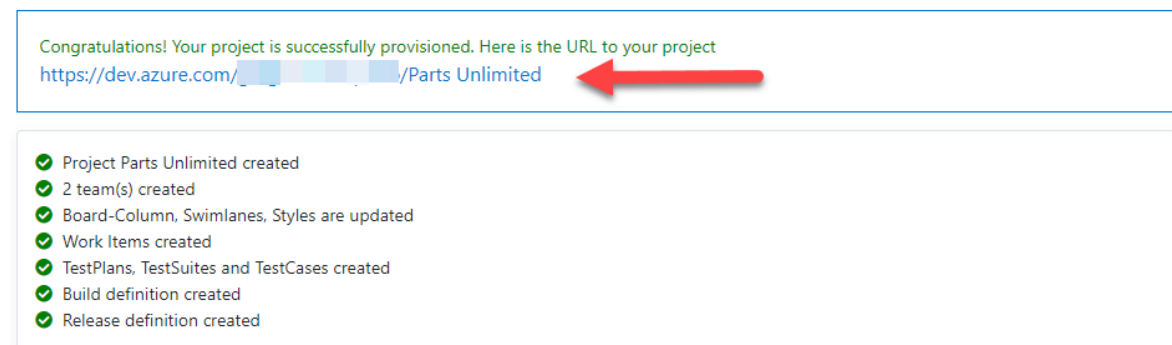
Create New Project

Select Organization :

New Project Name :

Selected Template :

1. Now that the Create New Project screen is completed, click **Create Project** to begin the project creation phase.



1. When the project is successfully completed, click the link provided to go to your team project within Azure DevOps.

- ✓ Note that because Azure DevOps was previously called VSTS (Visual Studio Team Services), some of the existing hands-on labs might refer to VSTS rather than Azure DevOps.



Module 1 Implementing Continuous Integration in an Azure DevOps Pipeline

Continuous Integration Overview

Lesson Overview

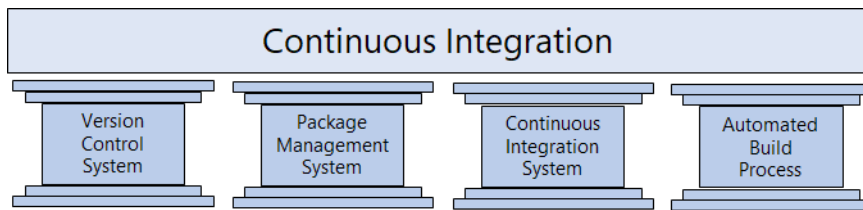
This lesson includes the following topics:

- Introduction to Continuous Integration
- The Four Pillars of Continuous Integration
- Benefits of Continuous Integration
- Continuous Implementation Challenges
- Implementing Continuous Integration in Azure DevOps
- Using Variables to Avoid Hard-coded Values
- Build Number Formatting and Status
- Build Authorizations Timeouts and Badges
- Configuring Build Retention
- Lab - Enabling Continuous Integration with Azure Pipelines

Video: Introduction to Continuous Integration



The Four Pillars of Continuous Integration



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**¹
- **Apache Subversion**²
- **Team Foundation Version Control**³

A **Package Management System** is used to install, uninstall and manage software packages.

- **NuGet**⁴
- **Node Package Manager**⁵
- **Chocolatey**⁶
- **HomeBrew**⁷
- **RPM**⁸

A **Continuous Integration System** merges all developer working copies to a shared mainline several times a day.

- **Azure DevOps**⁹
- **TeamCity**¹⁰
- **Jenkins**¹¹

An **Automated Build Process** creates a software build including compiling, packaging, and running automated tests.

- **Apache Ant**¹²
- **NAnt**¹³
- **Gradle**¹⁴

¹ <https://git-scm.com/>

² <https://subversion.apache.org/>

³ <https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/overview?view=vsts>

⁴ <https://www.nuget.org/>

⁵ <https://www.npmjs.com/>

⁶ <https://chocolatey.org/>

⁷ <https://brew.sh/>

⁸ <http://rpm.org/>

⁹ <https://azure.microsoft.com/en-us/services/devops>

¹⁰ <https://www.jetbrains.com/teamcity/>

¹¹ <https://jenkins.io/>

¹² <http://ant.apache.org/>

¹³ <http://nant.sourceforge.net/>

¹⁴ <https://gradle.org/>

✓ For each element, your team needs to select the specific platforms and tools they will use and you must ensure that you have established each pillar before proceeding.

Benefits of Continuous Integration

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 rapid feedback and early detection of problems (risk reduction)
- Better managing technical debt and conducting code analysis
- Reducing long, difficult, and bug-inducing merges
- Increasing confidence in codebase health long before production deployment

Key Benefit: Rapid Feedback for Code Quality

Possibly the most important benefit of continuous integration is rapid feedback to the developer. If the developer commits something and it breaks the code, he or she will know that almost immediately from the build, unit tests, and through other metrics. If successful integration is happening across the team, the developer is also going to know if their code change breaks something that another team member did to a different part of the code base. This process removes very long, difficult, and drawn out bug-inducing merges, which allows organizations to deliver in a very fast cadence.

✓ Continuous integration also enables tracking metrics to assess code quality over time, such as unit test pass rates, code that breaks frequently, code coverage trends, and code analysis. It can be used to provide information on what has been changed between builds for traceability benefits, as well as for introducing evidence of what teams do in order to have a global view of build results.

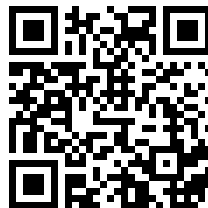
For more information, you can see:

What is Continuous Integration - <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration>

Discussion - CI Implementation Challenges

Have you tried to implement continuous integration in your organization? Were you successful? If you were successful, what lessons did you learn? If you were not successful, what were the challenges?

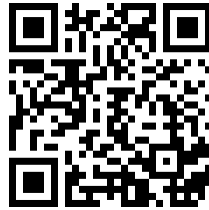
Video: Implementing Continuous Integration in Azure DevOps



Demonstration - Using Variables to Avoid Hard-coded Values

For more information, you can see:

Predefined Build Variables - <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables?view=vsts>



Build Number Formatting and Status

You may have noticed that in some demos, the build number was just an integer, yet in other demos, there is a formatted value that was based upon the date. This 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:

🏠 ... > Parts Unlimited-ASP.NET-CI

Tasks Variables Triggers **Options** Retention History | 📄 Save & queue ▾ ↶ Discard ☰ Summary

Build properties
Define general build pipeline settings

Description

Build number format ⓘ

\$(date:yyyyMMdd)\$(rev:.r)

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 will see in the next lesson that builds can be automatically triggered. This is a key capability required for continuous integration. But there are times that we might not want the build to run, even if it is triggered. This 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 that you can use the **Paused** setting to allow new builds to queue but to then hold off starting them.

For more information, you can see:

Build Pipeline Options - <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/options?view=vsts&tabs=yaml>

Build Authorizations Timeouts and Badges

Authorization and Timeouts

You can configure properties for the build job as shown below:

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

The authorization scope determines whether the build job is limited to accessing resources in the current project, or if it can access 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 is 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

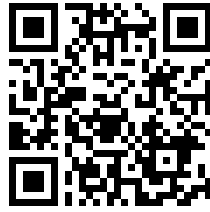
Some development teams like to show the state of the build on an external monitor or website. These settings provide a link to the image to use for that. Here is an example Azure Pipelines badge that has Succeeded.:



For more information, you can see:

Build Pipeline Options - <https://docs.microsoft.com/en-us/azure/devops/pipelines/build/options?view=vsts&tabs=yaml>

Video: Configuring Build Retention



Lab - Enabling Continuous Integration with Azure Pipelines

In this hands-on lab, **Enabling Continuous Integration with Azure Pipelines¹⁵**, you will learn how to configure continuous integration with Azure Pipelines.

You will perform the following tasks:

- Creating a basic build pipeline from a template
 - Tracking and reviewing a build
 - Invoking a continuous integration build
- ✓ Note that you must have already completed the Lab Environment Setup in the Welcome section.

¹⁵ <https://www.azuredevopslabs.com/labs/azuredevops/continuousintegration/>

Implementing a Build Strategy

Lesson Overview

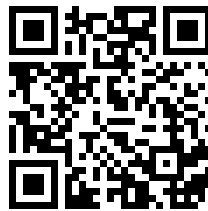
This lesson covers the following topics related to implementation of a build strategy:

- Automated Build Workflows
- Implementing Build Triggers
- Working with Hosted Agents
- Implementing a Hybrid Build Process
- Configuring Agent Demands
- Implementing Multi-Agent Builds
- Build-Related Tooling
- Creating a Jenkins Build Job and Triggering CI

Video: Automated Build Workflows



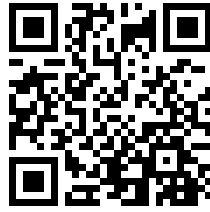
Video: Implementing Build Triggers



Video: Working with Hosted Agents



Video: Implementing a Hybrid Build Process



Configuring Agent Demands

Not all agents are the same. We've seen that they can be based on different operating systems, but they can also have different dependencies installed. To describe this, every agent has a set of capabilities configured as name-value pairs. The capabilities such as machine name and type of operating system that are automatically discovered, are referred to as **system capabilities**. The ones that you define are called **user capabilities**.

On the Agent Pools page (at the Organization level), when you select an agent, there is a tab for **Capabilities**. You can use it to see the available capabilities for an agent, and to configure user capabilities. For the self-hosted agent that I configured in the earlier demo, you can see the capabilities on that tab:

Requests **Capabilities**

USER CAPABILITIES

Shows information about user-defined capabilities supported by this host

[+ Add capability](#)

[Save changes](#) [Undo changes](#)

SYSTEM CAPABILITIES

Shows information about the capabilities provided by this host

Capability name	Capability value
Agent.ComputerName	GREGP50
Agent.HomeDirectory	C:\agent
Agent.Name	GREGP50
Agent.OS	Windows_NT
Agent.OSArchitecture	X64
Agent.OSVersion	10.0.17134
Agent.Version	2.141.2
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\Greg\AppData\Roaming
AzurePS	5.7.0
bower	C:\Users\Greg\AppData\Roaming\npm\bower.cmd
Cmd	C:\WINDOWS\system32\cmd.exe
CommonProgramFiles	C:\Program Files\Common Files
CommonProgramFiles(x86)	C:\Program Files (x86)\Common Files
CommonProgramW6432	C:\Program Files\Common Files
COMPUTERNAME	GREGP50

When you configure a build pipeline, as well as the agent pool to use, on the **Options** tab, you can specify certain demands that the agent must meet.

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 above image, the HasPaymentService is required in the collection of capabilities. As well as an **exists** condition, you can choose that a capability **equals** a specific value.

For more information, you can see:

Capabilities - <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=vsts#capabilities>

Implementing Multi-Agent Builds

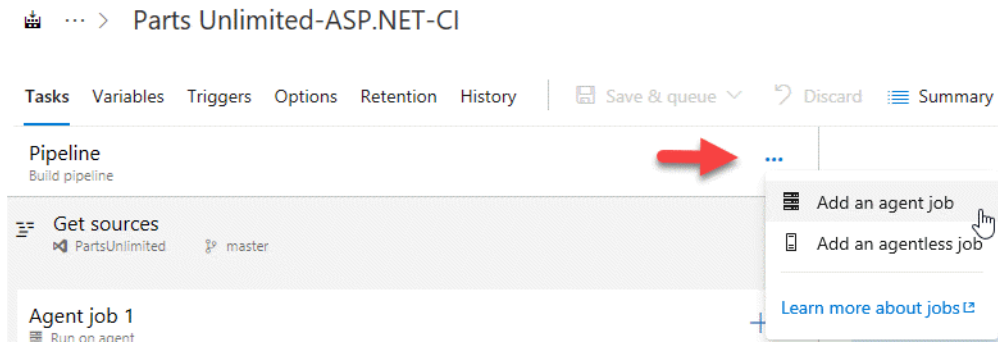
You can use multiple build agents to support multiple build machines, either to distribute the load, to run builds in parallel, or to use different agent capabilities. As an example, components of an application might require different incompatible versions of a library or dependency.

Multiple Jobs in a Pipeline

Adding multiple jobs to a pipeline lets you:

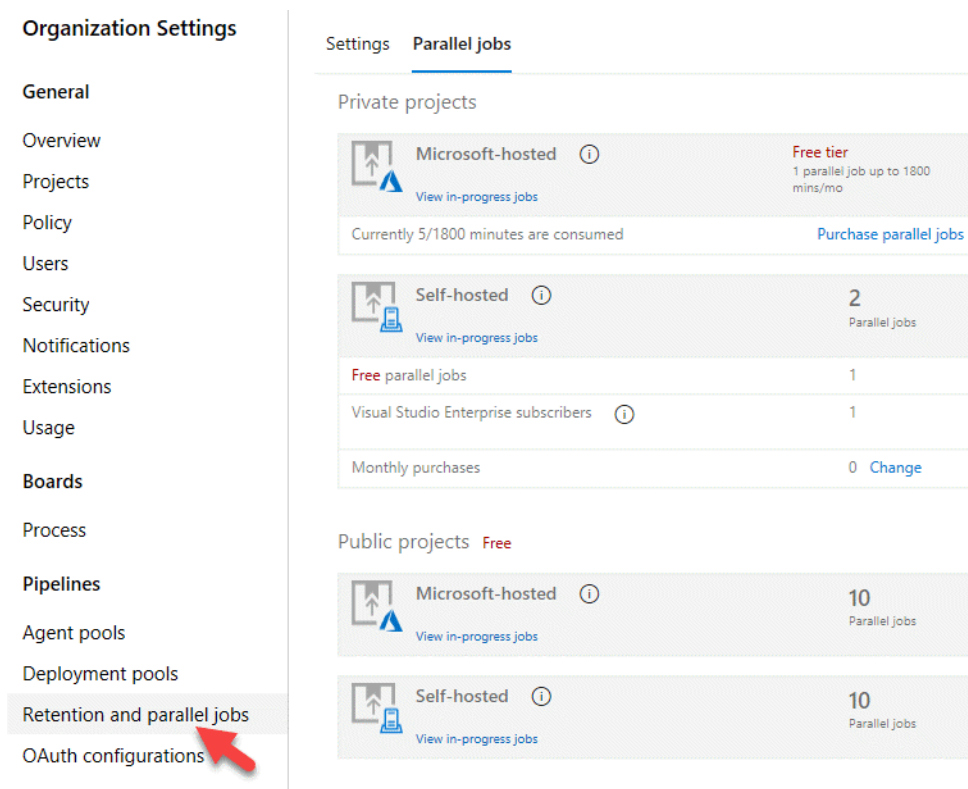
- Break your pipeline into sections that need different agent pools, or self-hosted agents
- Publish artifacts in one job and consume them in one or more subsequent jobs
- Build faster by running multiple jobs in parallel
- Enable conditional execution of tasks

To add another agent job to an existing pipeline, click on the ellipsis and choose as shown in this image:



Parallel Jobs

At the Organization level, you can configure the number of parallel jobs that are made available.



The free tier allows for one parallel job of up to 1800 minutes per month. The self-hosted agents has higher levels.

✓ 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 runs on an agent. If there aren't enough parallel jobs available for your organization, the jobs will be queued and run sequentially.

Discussion - Build-Related Tooling

Azure DevOps can be integrated with a wide range of existing tooling that is 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?

Lab - Creating a Jenkins Build Job and Triggering CI

There are two ways to integrate Jenkins with Azure Pipelines:

- One way is to run CI jobs in Jenkins separately. This involves configuration of a CI pipeline in Jenkins and a web hook in Azure DevOps that invokes the CI process when source code is pushed to a repository or a branch.
- The alternate way is to wrap a Jenkins CI job inside an Azure pipeline. In this approach, a build definition will be configured in Azure Pipelines to use the Jenkins tasks to invoke a CI job in Jenkins, download and publish the artifacts produced by Jenkins

In this hands-on lab, **Integrating Jenkins CI with Azure Pipelines**¹⁶, you will try both approaches and learn how to:

- Provision Jenkins on Azure VM using the Jenkins template available on the Azure Marketplace
 - Configure Jenkins to work with Maven and Azure DevOps
 - Create a build job in Jenkins
 - Configure Azure Pipeline to integrate with Jenkins
 - Configure a CD pipeline in Azure Pipelines to deploy the build artifacts
- ✓ This is a lengthy lab, just take your time and work through the steps.

¹⁶ <https://www.azuredevopslabs.com/labs/vstsextend/jenkins/>

Module 1 Review Questions

Module 1 Review Questions

Pillars of Integration

Name and discuss the four pillars of continuous integration.

Click for suggested answer ↓

The four pillars of continuous integration are: a Version Control System, Packet Management System, Continuous Integration System, and an Automated Build Process.

Build Numbers

The build numbers that Azure DevOps generates for you are currently integers. You would prefer the build to include the date. How would you change this?

Click for suggested answer ↓

The Build Options tab for a build includes the format for build number generation.

Configuration Changes

You want to take your build server offline to make a configuration change. You want it to complete any build that it is currently processing, but you want to queue any new build requests. What should you do?

Click for suggested answer ↓

You should pause the build. A paused build will not start new builds and will queue any new build requests.

Build Times

You want to set a maximum time that builds can run for. Builds should not run for more than 5 minutes. What configuration change should you make?

Click for suggested answer ↓

You should change the build job timeout setting to 5 minutes. A blank value means unlimited.

Enabling Continuous Integration

The hands-on lab for Creating a Jenkins Build Job and Triggering CI described two methods that could be used to enable continuous integration for Jenkins. What were the two methods?

Click for suggested answer ↓

In the hands-on lab, the two methods described were triggering the CI via a service hook in Azure DevOps, and wrapping the Jenkins job within Azure Pipelines.

Module 2 Managing Code Quality and Security Policies

Managing Code Quality

Lesson Overview

This lesson covers the following topics related to managing code quality:

- Code Quality Defined
- Sources and Impacts of Technical Debt
- Using Automated Testing to Measure and Monitor Technical Debt
- Configuring SonarCloud in a Build Pipeline
- Reviewing SonarCloud Results and Resolving Issues
- Integrating Other Code Quality Tools
- Code Quality Tooling
- Managing Technical Debt with Azure DevOps and SonarCloud

Video: Code Quality Defined



Sources and Impacts of Technical Debt

Technical Debt is a term that describes the future penalty that you incur today by making easy or quick choices in software development practices, rather than taking harder or longer choices that are more appropriate longer-term. Over time, it accrues 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 orient design principles.
- Monolithic classes and code libraries.
- Poorly envisioned use of technology, architecture and approach. (Forgetting that all attributes of the system, affecting maintenance, user experience, scalability, and others, need to be considered).
- Over-engineering code (adding or creating code that is not needed, adding custom code when existing libraries are sufficient, or creating layers or components that are not 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.

✓ Over time, technical debt must be paid back. Otherwise, the team's ability to fix issues, and to implement new features and enhancements will take longer and longer, and eventually become cost-prohibitive.

Using Automated Testing to Measure Technical Debt

We have seen that technical debt adds a set of problems during development and makes it much more difficult to add additional 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 to 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 needed 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 cannot 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 take a look at one of the common tools that is used to assess the debt: SonarCloud. (The original on-premises version was SonarQube).

There are other tools available and we will discuss a few of them. Later, in the next hands-on lab, you will see how to configure your Azure Build definitions to use SonarCloud, how to 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, you can see:

SonarCloud - <https://sonarcloud.io/about>

Video: Configuring SonarCloud in a Build Pipeline



Video: Reviewing SonarCloud Results and Resolving Issues



Integrating Other Code Quality Tools

There are many tools that 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 that a developer has added during a recent development period, typically in the last hour. With this information, the developer might be able to make the required corrections before ever committing the code. NDepend lets you create code rules that are expressed as C# LINQ queries but it has a large number of 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, and can be set to automatically fail builds when code quality issues are found. 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 |  Refreshquality 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 - <https://www.ndepend.com>

Visual Studio marketplace - <https://marketplace.visualstudio.com/items?itemName=ndepend.ndependextension&targetId=2ec491f3-0a97-4e53-bfef-20bf80c7e1ea>

Resharper Code Quality Analysis - <https://marketplace.visualstudio.com/items?itemName=alan-wales.resharper-code-analysis>

Discussion - Code Quality Tooling

Azure DevOps can be integrated with a wide range of existing tooling that is used for checking code quality during builds. Which code quality tools do you currently use (if any)? What do you like or don't like about the tools?

Lab - Managing Technical Debt with Azure DevOps and SonarCloud

In this hands-on lab, **Managing Technical Debt with Azure DevOps and SonarCloud¹**, you will learn how to manage and report on technical debt using SonarCloud integration with Azure DevOps.

You will perform the following tasks:

- Integrate SonarCloud with Azure DevOps and run an analysis
 - Analyze the results
 - Configure a quality profile to control the rule set used for analyzing your project
- ✓ Note that you must have already completed the Lab Environment Setup in the Welcome section.

¹ <https://www.azuredevopslabs.com/labs/azuredevops/sonarcloud/>

Managing Security Policies

Lesson Overview

This lesson covers the following topics related to managing security policies:

- Open Source Licensing Challenges
- Avoiding the OWASP Top Ten
- Detecting Open Source Issues with WhiteSource Bolt
- Integrating Other Security Policy Tooling
- Security Policy Tooling
- Checking Vulnerabilities using WhiteSource Bolt and Azure DevOps

Video - Open Source Licensing Challenges

For more information, you can see:

Common Vulnerabilities and Exposures - <https://cve.mitre.org/about/>



Video - Avoiding the OWASP Top Ten

For more information, you can see:

OWASP Top Ten - https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project



Video: Detecting Open Source Issues with WhiteSource Bolt



Integrating Other Security Policy Tooling

Azure DevOps can be integrated with a wide range of other tools that can help in checking code for vulnerabilities. Four common tools are:

Micro Focus Fortify²

Fortify provides build tasks that can be used in Azure DevOps continuous integration builds to identify vulnerabilities in source code. It offers two styles of analysis.

- **Fortify Static Code Analyzer** (SCA) searches for violations of security-specific coding rules and guidelines. It works in a variety of languages.
- **Fortify on Demand** is a service for checking application security. The outcomes of an SCA scan are audited by a team of security experts, including the use of Fortify WebInspect for automated dynamic scanning.

Checkmarx CxSAST³

Checkmarx is a solution for Static Source Code Analysis (SAST) and Open Source Analysis (OSA) designed for identifying, tracking and fixing technical and logical security flaws.

It is designed to integrated into Azure DevOps pipelines and allows for early detection and mitigation of crucial security flaws. To improve performance, it is capable of incremental scanning (ie: just checking the code recently altered or introduced).

BinSkim⁴

BinSkim is a static analysis tool that scans binary files. In particular, it checks that the executable produced (ie: a Windows PE formatted file) has opted into all of the binary mitigations offered by the Windows Platform, including:

- SafeSEH is enabled for safe exception handling
- ASLR is enabled so that memory is not laid out in a predictable fashion
- Stack Protection is enabled to prevent overflow

BinSkim replaces an earlier Microsoft tool called BinScope.

OWASP Zed Attack Proxy Scan⁵

The OWASP Zed Attack Proxy (known as **OWASP ZAP**) Scan is an open-source web application security scanner that is intended for users with all levels of security knowledge. It can be used by professional penetration testers.

Discussion - Security Policy Tooling

Azure DevOps can be integrated with a wide range of existing tooling that is used for checking security policy during builds.

Which security policy tools do you currently use (if any)? What do you like or don't like about them?

² <https://marketplace.visualstudio.com/items?itemName=fortifyvsts.hpe-security-fortify-vsts>

³ <https://marketplace.visualstudio.com/items?itemName=checkmarx.cxsast>

⁴ <https://blogs.msdn.microsoft.com/secdevblog/2016/08/17/introducing-binskim/>

⁵ <https://marketplace.visualstudio.com/items?itemName=kasunkodagoda.owasp-zap-scan>

Lab - Checking Vulnerabilities using WhiteSource Bolt and Azure DevOps

In this hands-on lab, **Checking Vulnerabilities using WhiteSource Bolt with Visual Studio Team Services⁶**, you will learn how to check for open source vulnerabilities using WhiteSource Bolt in conjunction with Azure DevOps.

You will Learn to do the following:

- Integrate WhiteSource Bolt with you Azure DevOps build process
 - Detect and remedy vulnerable open source components
 - Generate comprehensive open source inventory reports per project or build
 - Enforce open source license compliance, including licenses for dependencies
 - Identify outdated open source libraries with recommendations to update
- ✓ Note that you must have already completed the Lab Environment Setup in the Welcome section. Also, that at the time of writing, this lab refers to VSTS instead of Azure DevOps.

⁶ <https://www.azuredevopslabs.com/labs/vstsextend/WhiteSource/>

Module 2 Review Questions

Module 2 Review Questions

Testing Tool

You want to run a penetration test against your application. Which tool could you use?

Click for suggested answer ↓

OWASP ZAP. OWASP ZAP is designed to run penetration testing against applications. Bolt is used to analyze open source library usage. The two Sonar products are for code quality and code coverage analysis.

Code Smells

What is code smells? Give an example of a code smell.

Click for suggested answer ↓

Code smells are characteristics in your code that could possibly be a problem. Code smells hint at deeper problems in the design or implementation of the code. For example, code that works but contains many literal values or duplicated code.

Auditing Tool

You are using Azure Repos for your application source code repository. You want to create an audit of open source libraries that you have used. Which tool could you use?

Click for suggested answer ↓

WhiteSource Bolt is used to analyze open source library usage. OWASP ZAP is designed to run penetration testing against applications. The two Sonar products are for code quality and code coverage analysis.

High Quality Code

Name three attributes of high quality code.

Click for suggested answer ↓

High quality code should have well-defined interfaces. It should be clear and easy to read so self-documenting is desirable, as is short (not long) method bodies.

High Quality Code

You are using Azure Repos for your application source code repository. You want to perform code quality checks. Which tool could you use?

Click for suggested answer ↓

SonarCloud is the cloud-based version of the original SonarQube, and would be best for working with code in Azure Repos.

Module 3 Implementing a Container Strategy

Implementing a Container Build Strategy

Lesson Overview

This lesson covers the following topics related to implementing a build container strategy:

- Overview of Containers
- Containers vs Virtual Machines
- Docker Containers and Development
- Microservices and Containers
- Azure Container-Related Services
- Dockerfile Core Concepts
- Creating Multi-Stage Builds
- Creating an Azure Container Registry
- Adding Docker Support to an Existing Application
- Additional Container-Related Resources
- Modernizing an Existing .NET Application with Azure and Docker Images

Video: Overview of Containers



Discussion - Containers vs Virtual Machines

In your development environment, do you currently use virtualization of any type? Do you prefer to deploy containers or entire virtual machines?

Docker Containers and Development



Docker is a software containerization platform with a common toolset, packaging model, and deployment mechanism, which greatly simplifies containerization and distribution of applications that can be run anywhere. This ubiquitous technology not only simplifies management by offering the same management commands against any host, it also creates a unique opportunity for seamless DevOps.

From a developer's desktop to a testing machine, to a set of production machines, a Docker image can be created that will deploy identically across any environment in seconds. This is a massive and growing ecosystem of applications packaged in Docker containers, with DockerHub, the public containerized-application registry that Docker maintains, currently publishing more than 180,000 applications in the public community repository. Additionally, to guarantee the packaging format remains universal, Docker organized the Open Container Initiative (OCI), aiming to ensure container packaging remains an open and foundation-led format.

As an example of the power of containers, a SQL Server Linux instance can be deployed using a Docker image in seconds.

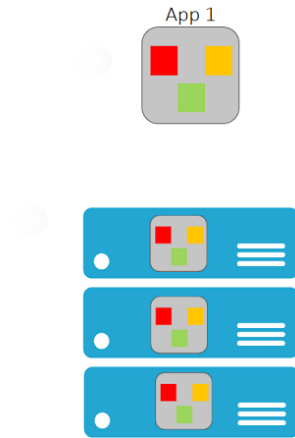
For more information, you can see:

Docker Ebook, Docker for the Virtualization Admin - <https://goto.docker.com/docker-for-the-virtualization-admin.html>

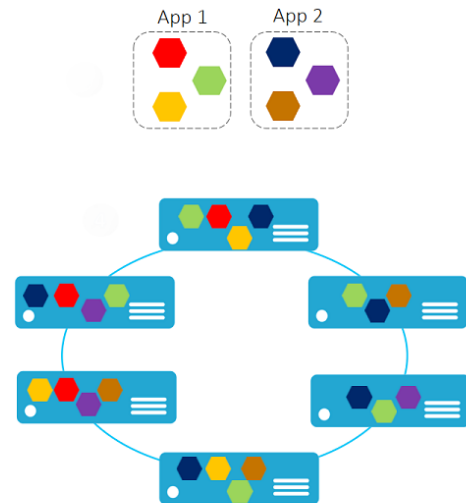
Mark Russinovich blog post on Containers: Docker, Windows, and Trends- <https://azure.microsoft.com/en-us/blog/containers-docker-windows-and-trends/>

Microservices and Containers

Monolithic application approach



Microservices application approach



The most immediately lucrative use for containers has been focused on simplifying DevOps with easy developer-to-test-to-production flows for services deployed in the cloud or on-premises. But there is another fast-growing scenario where containers are becoming very compelling.

Microservices is an approach to application development where every part of the application is deployed as a fully self-contained component, called a microservice, that can be individually scaled and updated.

Example Scenario

Imagine that you are part of a software house that produces a large monolithic financial management application that you are migrating to a series of microservices. The existing application would include the code to update the general ledger for each transaction, and it would have this code in many places throughout the application. If the schema of the general ledger transactions table is modified, this would require changes throughout the application.

By comparison, the application could be modified to make a notification that a transaction has occurred. Any microservice that is interested in the transactions could subscribe. In particular, a separate general ledger microservice could subscribe to the transaction notifications, and then perform the general ledger related functionality. If the schema of the table that holds the general ledger transactions is modified, only the general ledger microservice should need to be updated.

If a particular client organization wants to run the application and not use the general ledger, that service could just be disabled. No other changes to the code would be required.

Scale

In a dev/test environment on a single system, while you might have a single instance of each microservice, in production you might scale out to different numbers of instances across a cluster of servers depending on their resource demands as customer request levels rise and fall. If different teams produce them, the teams can also independently update them. Microservices is not a new approach to programming, nor is it tied explicitly to containers, but the benefits of Docker containers are magnified when applied to a complex microservice-based application. Agility means that a microservice can quickly scale

out to meet increased load, the namespace and resource isolation of containers prevents one microservice instance from interfering with others, and use of the Docker packaging format and APIs unlocks the Docker ecosystem for the microservice developer and application operator. With a good microservice architecture, customers can solve the management, deployment, orchestration and patching needs of a container-based service with reduced risk of availability loss while maintaining high agility.

Azure Container-Related Services

Azure provides a wide range of services that help you to work with containers. These are the key services that are involved:

Azure Container Instances (ACI)¹

Running your workloads in Azure Container Instances (ACI), allows you to focus on creating your applications rather than focusing on provisioning and management of the infrastructure that will run the applications.

ACIs are simple and fast to deploy, and when you are using them, you gain the security of hypervisor isolation for each container group. This ensures that your containers aren't sharing an operating system kernel with other containers.

Azure Kubernetes Service (AKS)²

Kubernetes has quickly become the de facto standard for container orchestration. This service lets you easily deploy and manage Kubernetes, to scale and run applications, while maintaining strong overall security.

This service started life as Azure Container Services (ACS) and at release supported Docker Swarm and Mesos/Mesosphere DC/OS for managing orchestrations. These original ACS workloads are still supported in Azure but Kubernetes support was added. This 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)³

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 types of container deployments, including DC/OS, Docker Swarm, Kubernetes are supported and the registry is integrated with other Azure services such as the App Service, Batch, Service Fabric and others.

Importantly, this allows your DevOps team to manage the configuration of apps without being tied to the configuration of the target hosting environment.

Azure Service Fabric⁴

Azure Service Fabric allows you to build and operate always-on, scalable, distributed apps. It simplifies the development of microservice-based applications and their life cycle management including rolling updates with rollback, partitioning, and placement constraints. It can host and orchestrate containers, including stateful containers.

Azure App Service⁵

¹ <https://azure.microsoft.com/en-us/services/container-instances/>

² <https://azure.microsoft.com/en-us/services/kubernetes-service/>

³ <https://azure.microsoft.com/en-us/services/container-registry/>

⁴ <https://azure.microsoft.com/en-us/services/service-fabric/>

⁵ <https://azure.microsoft.com/en-us/services/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 options for auto-scaling and load balancing and is easy to integrate with Azure DevOps.

Dockerfile Core Concepts

Dockerfiles are text files that contain the commands needed by **docker build** to assemble an image.

Here is an example of a very basic Dockerfile:

```
FROM ubuntu
LABEL maintainer="greglow@contoso.com"
ADD appsetup /
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
CMD ["echo", "Hello World from within the container"]
```

Note the following in this example.

The first line refers to the parent image that this new image will be based upon. Generally, all images will be based off 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 into 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. This is equivalent to having written this instead:

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

The RUN command is run when the image is being created by **docker build**. It is generally 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 ie: it is run after container creation.

For more information, you can see:

Dockerfile reference - <https://docs.docker.com/engine/reference/builder/>

Multiple Stage Builds

It's important when building images, to keep the image size as small as possible. Every additional instruction that is added to the Dockerfile adds what is referred to as a **layer**. Each layer often contains artifacts that are not needed in the final image and should be cleaned up before moving to the next layer. Doing this has been tricky.

Multiple Stage Builds

Docker 17.05 added a new feature that allowed the creation of multi-stage builds. This helps with being able to optimize the files, improves their readability, and makes them easier to maintain.

Here is an example of a multi-stage file that was created by Visual Studio 2017:

```
FROM microsoft/dotnet:2.1-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 6636
EXPOSE 44320

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY ["WebApplication1/WebApplication1.csproj", "WebApplication1/"]
RUN dotnet restore "WebApplication1/WebApplication1.csproj"
COPY . .
WORKDIR "/src/WebApplication1"
RUN dotnet build "WebApplication1.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "WebApplication1.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

With multi-stage builds, you use multiple FROM statements in your Dockerfile. Each FROM instruction starts a new stage. The stages are numbered in order, starting with stage 0. To make the file easier to maintain without needing to constantly change numbers that reference, note how each stage has been named (or aliased) by using an **AS** clause.

Each FROM instruction can have a different parent (ie: base). This allows the developer to control what is copied from one stage to another, and avoids the need for intermediate images.

Another advantage of named stages is that they are easier to refer to in external commands. For example, not all stages need to be built each time. You can see that in the following Docker CLI command:

```
$ docker build --target publish -t gregsimages/popkorn:latest .
```

The **--target** option tells **docker build** that it needs to create an image up to the target of publish, which was one of the named stages.

✓ The next topic covers best practices for multiple stage builds.

Considerations for Multiple Stage Builds

Adopt Container Modularity

Try to avoid creating overly complex container images that couple together a number of 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 being able to reuse 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 with minimizing image sizes, it is also important avoid including packages that you suspect might be needed but aren't yet sure if they are needed. Only include them when they are needed.

Choose an Appropriate Base

While optimizing the contents of your Dockerfiles is important, it is also very important to choose the appropriate parent (base) image. Start with an image that only contains packages that are required.

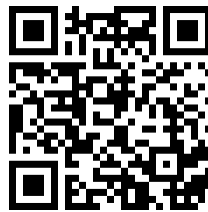
Avoid Including Application Data

While application data can be stored in the container, doing this will make your images larger. You should 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, you can see:

Use multiple stage builds - <https://docs.docker.com/develop/develop-images/multistage-build/>

Video: Create an Azure Container Registry



Video: Add Docker Support to an Existing Application



Lab - Existing .NET Applications with Azure and Docker Images

In this hands-on lab, **Modernizing your existing ASP.NET Apps with Azure**⁶, you will learn how to modernize an existing ASP.NET application with migration to Docker images managed by the Azure Container Registry.

You will perform the following tasks:

- Migrate the LocalDB to SQL Server in Azure
- Using the Docker tools in Visual Studio 2017, add Docker support for the application
- Publish Docker Images to Azure Container Registry (ACR)

⁶ <https://www.azuredevopslabs.com/labs/vstxextend/aspnetmodernize/>

- Push the new Docker images from ACR to Azure Container Instances (ACI)

Module 3 Review Questions

Module 3 Review Questions

Multiple Stage Builds

You are reviewing an existing Dockerfile. How would you know if it's a multi-stage Dockerfile?

Click for suggested answer ↓

Multi-stage Docker files are characterized by containing more than one starting point provided as FROM instructions.

Multi-stage Build Design

You are designing a multi-stage Dockerfile. How can one stage refer to another stage within the Dockerfile?

Click for suggested answer ↓

The FROM clause in a multi-stage Dockerfile can contain an alias via an AS clause. The stages can refer to each other by number or by the alias names.

Dockerfile Format

What is the line continuation character in Dockerfiles?

Click for suggested answer ↓

Lines can be broken and continued on the next line of a Dockerfile by using the backslash character.

Orchestration Styles

You are using Azure to manage your containers. Which container orchestration styles are supported?

Click for suggested answer ↓

Swarm, DC/OS, and AKS are supported.

Open Container Initiative

When the Open Container Initiative defined a standard container image file format, which format did they choose as a starting point?

Click for suggested answer ↓

The OCI used the Docker format as a starting point.