Project Name: **Azure DevOps Report**

Team members: **Devavrat Kalam(dk2792)**
Saurabh Gaikwad(sg4382)

What is DevOps

DevOps is a cluster of multiple methodologies and practices. To understand DevOps lets look in the past software development practices. In most cases, companies have developer teams and operations teams separately. The developer team is responsible for creating software whereas the Operation team tests the software and maintains them. This process created many issues such as different software dependencies on different computers, slow collaboration among teams resulting in slow deployment, etc.. DevOps targets these problems. DevOps allows operations and development engineers to participate together in the entire service lifecycle, from design through the development process to production support.

Goals of DevOps

As we discussed so far, DevOps try to solve many issues associated with the traditional software development life cycle.

DevOps key features include,

- **Reliability**: It ensures the quality of application updates and infrastructure changes so we can reliably deliver changes at a more rapid pace while maintaining a positive experience for end-users.
- **Scale**: Operate and manage infrastructure and development processes at scale. We can perform automation and consistency to help in managing complex or changing systems efficiently and with reduced risk.
- **Improve Development Frequency:** Increase the frequency and pace of releases so we can innovate and improve the product faster. The quicker we can release new features and fix bugs, the faster you can respond to the customer needs and build a competitive advantage. Continuous integration(CI) and continuous delivery(CD) are practices that automate the software release process, from build to deploy.
- **Lower Failure Rate of New Releases**: All the changes made in code and pushed to deployment undergo various tests. If any of the tests fail, changes are rollbacked and the current version is not affected.
- **Improve Collaboration**: Developers and operations teams collaborate closely, share many responsibilities, and combine their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).
- **Security**: We can adopt the DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and policy as code, you can define and then track compliance at scale.

Azure DevOps Services

Azure DevOps is a cloud provider that allows developers to support teams to plan work, collaborate on code development, build and deploy applications. Developers can use the cloud using Azure DevOps Services or on-premises using Azure DevOps Server.

Azure services include,

- **Azure Repos** - It provides Git repositories or Team Foundation Version Control for source control of your code.
- **Azure Pipelines** - It provides build and release services to support continuous integration and delivery for apps.
- **Azure Boards** - It delivers a suite of Agile tools to support planning and tracking work, code defects, and issues using Kanban and Scrum methods.
- **Azure Test Plans** - It provides several tools to test your apps, including manual/exploratory testing and continuous testing.
- **Azure Artifacts** - It allows teams to share Maven, npm, and NuGet packages from public and private sources and integrate package sharing into your CI/CD pipelines.

For this presentation, we will be discussing Azure DevOps Pipelines, Artifacts, Boards and Repos.

Azure Pipelines

Azure Pipelines is used to build, test, and deploy any application with CI/CD that works with any language, platform, and cloud. We can also connect the pipelines to GitHub or any other Git provider and deploy continuously. Once it's linked with a Git provider, It uses practices of continuous integration(CI) and continuous delivery(CD) to test that each change is functional and safe to be deployed. We can also monitor and log changes to help in staying informed of performance in real-time.

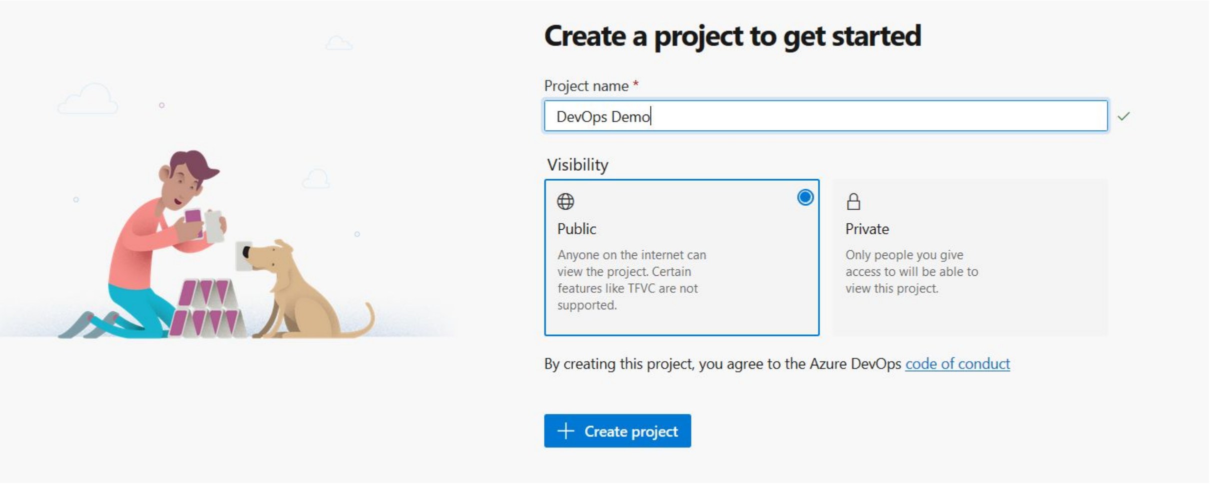
Key features of Azure Pipeline

- Azure Pipelines has developer support for any programming language or platform making it very easy to set up and run.
- We can deploy changes to different targets i.e. container registry, virtual machines, cloud targets, etc. at the same time.
- We can create different builds on different operating systems like Windows, Linux or Mac.
- Azure pipelines integrate with Github allowing easy imports and changes to pipeline.
- Azure pipelines have deep support for open-source projects.

DEMO: Creating your first pipeline

1. First, create an Azure account through this link. Azure provides $200 credit for their services and 1-year support.
2. Create an Organization from this link. An Organization is nothing but your personal space where you and your team can collaborate on projects using Azure services.
3. Once we start the Organization, we get the following page. Name the project to create and select whether to make it public or private. In public projects, Team Foundation

Version Control(TFVC) features won't be available. We can select which version control to use between Git or TFVC. We can also select the work item process for the project such as Agile, Scrum, etc. Finally, click Create project.



Create a project to get started

Project name *

DevOps Demo

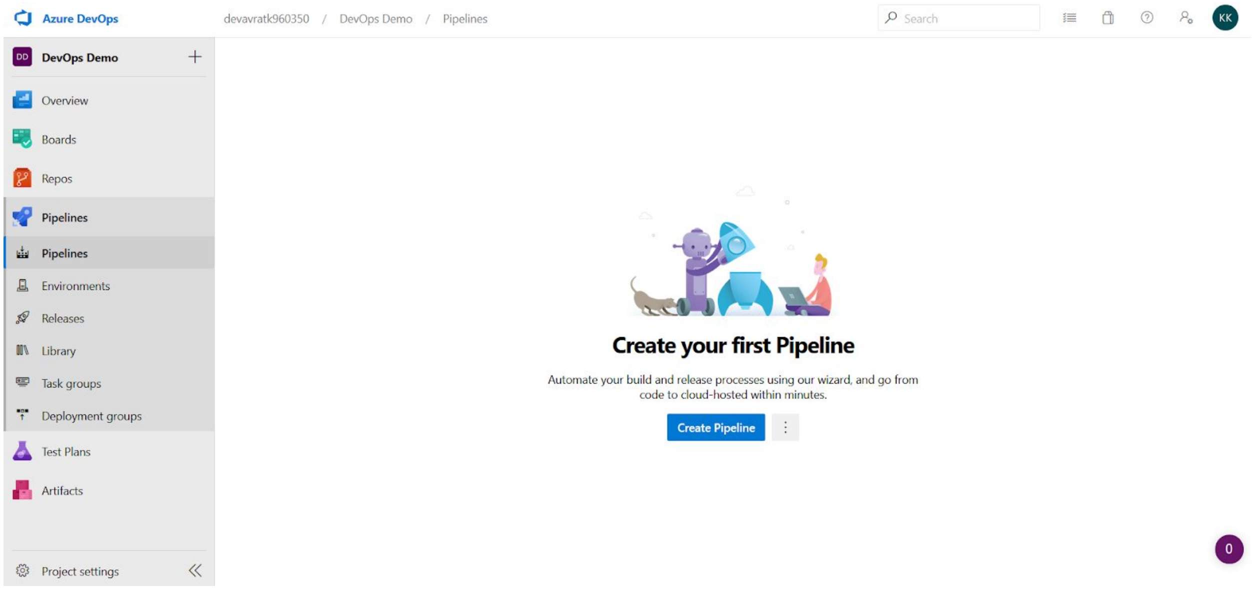
Visibility

● Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

Private
Only people you give access to will be able to view this project.

By creating this project, you agree to the Azure DevOps code of conduct

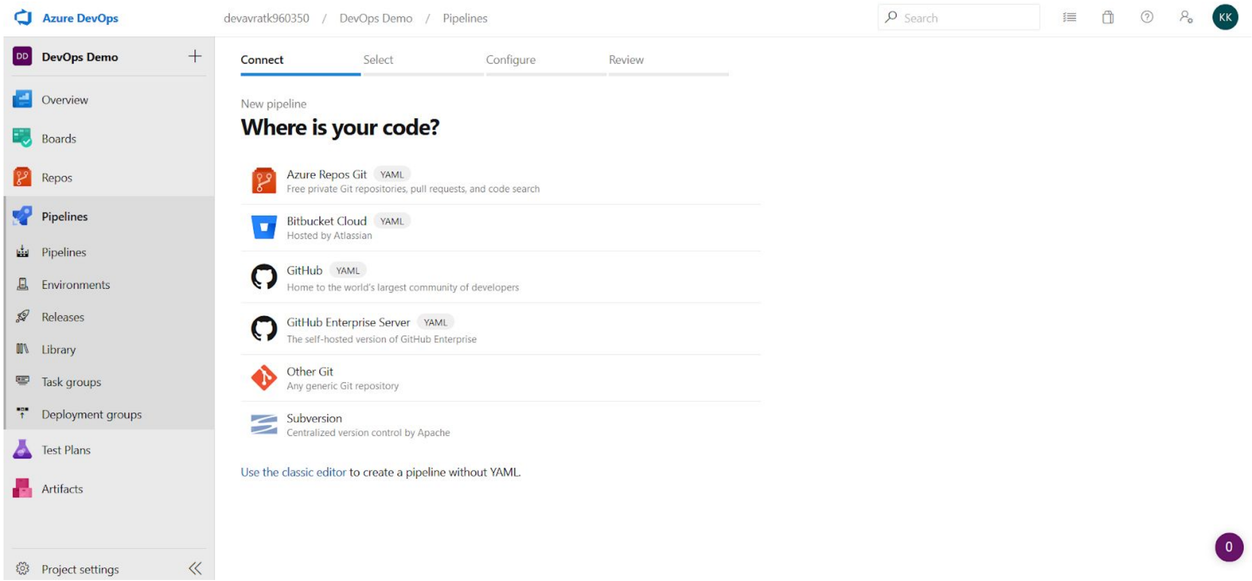
4. Once Azure sets up the workspace, we can see all the services provided by azure such as Pipeline, Repos, Boards, etc. Select "Pipelines" and click "Create Pipeline".



Create your first Pipeline

Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.

5. Select the option where your code is located. For this demo, I am using Github as the source repository for my project. Select the repository we want to use from Github. A YAML file will automatically be generated based on the project and its relevant dependencies. Click on Run to create the pipeline.



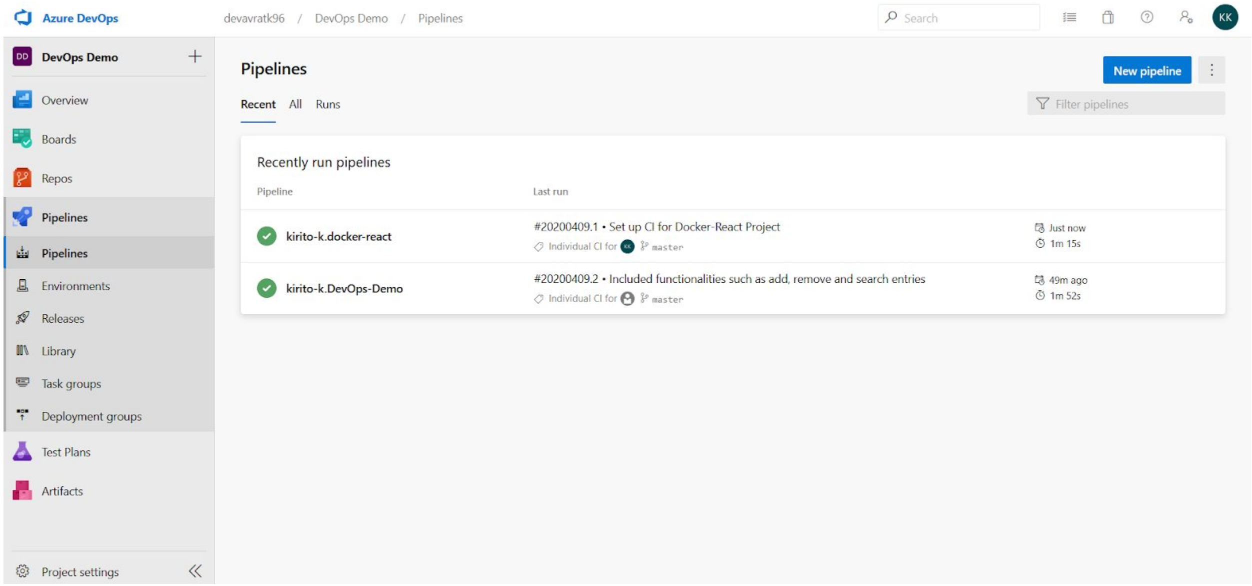
New pipeline

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

Use the classic editor to create a pipeline without YAML.

6. After you imported everything, we can see the list of projects for which we have built Pipelines.

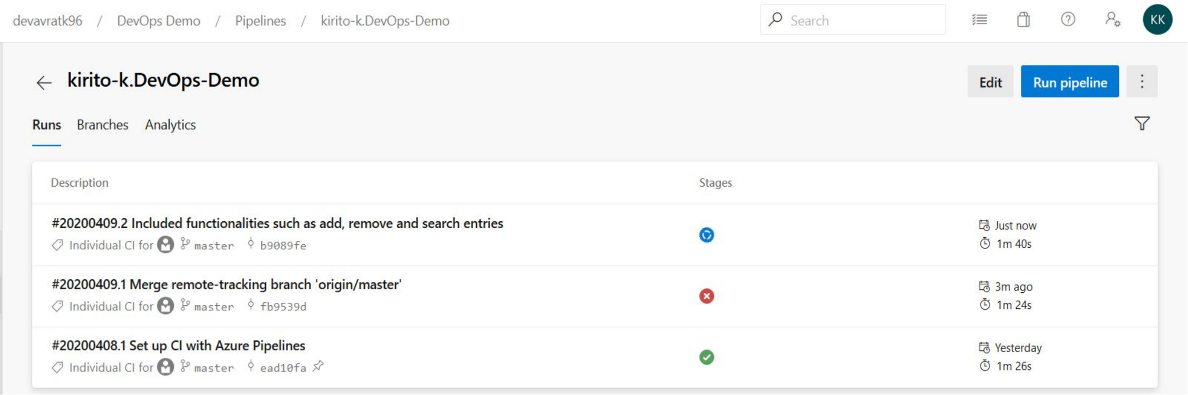


Pipelines

Recently run pipelines

Pipeline	Last run	
kirito-k.docker-react	#20200409.1 • Set up CI for Docker-React Project Individual CI for KK master	Just now 1m 15s
kirito-k.DevOps-Demo	#20200409.2 • Included functionalities such as add, remove and search entries Individual CI for master	49m ago 1m 52s

7. Select a project to see different records related to that project's pipelines such as the number of runs and various Analytics which includes pipeline's build time and success rate etc.



devavratk96 / DevOps Demo / Pipelines / kirito-k.DevOps-Demo

Search

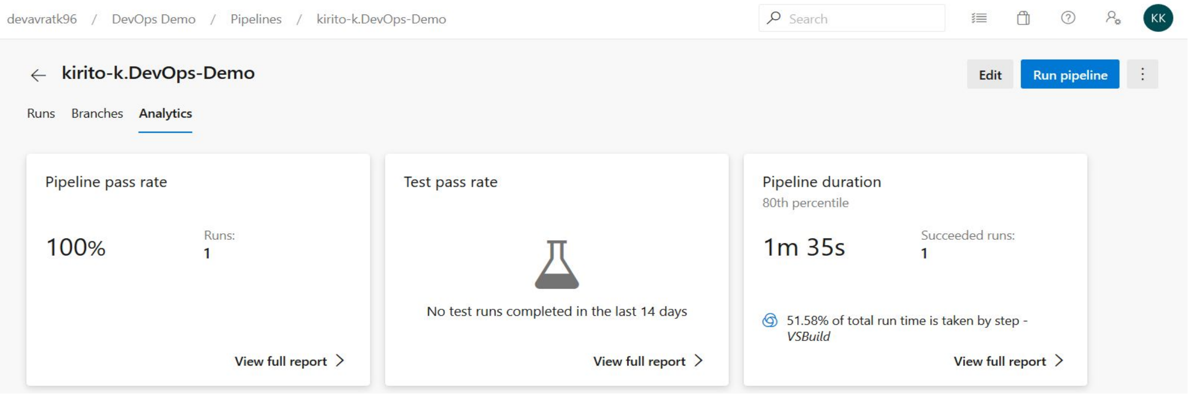
KK

kirito-k.DevOps-Demo

Edit Run pipeline

Runs Branches Analytics

Description	Stages	
#20200409.2 Included functionalities such as add, remove and search entries Individual CI for master b9089fe		Just now 1m 40s
#20200409.1 Merge remote-tracking branch 'origin/master' Individual CI for master fb9539d		3m ago 1m 24s
#20200408.1 Set up CI with Azure Pipelines Individual CI for master ead10fa		Yesterday 1m 26s



devavratk96 / DevOps Demo / Pipelines / kirito-k.DevOps-Demo

Search

KK

kirito-k.DevOps-Demo

Edit Run pipeline

Runs Branches Analytics

Pipeline pass rate

100% Runs: 1

View full report

Test pass rate

No test runs completed in the last 14 days

View full report

Pipeline duration

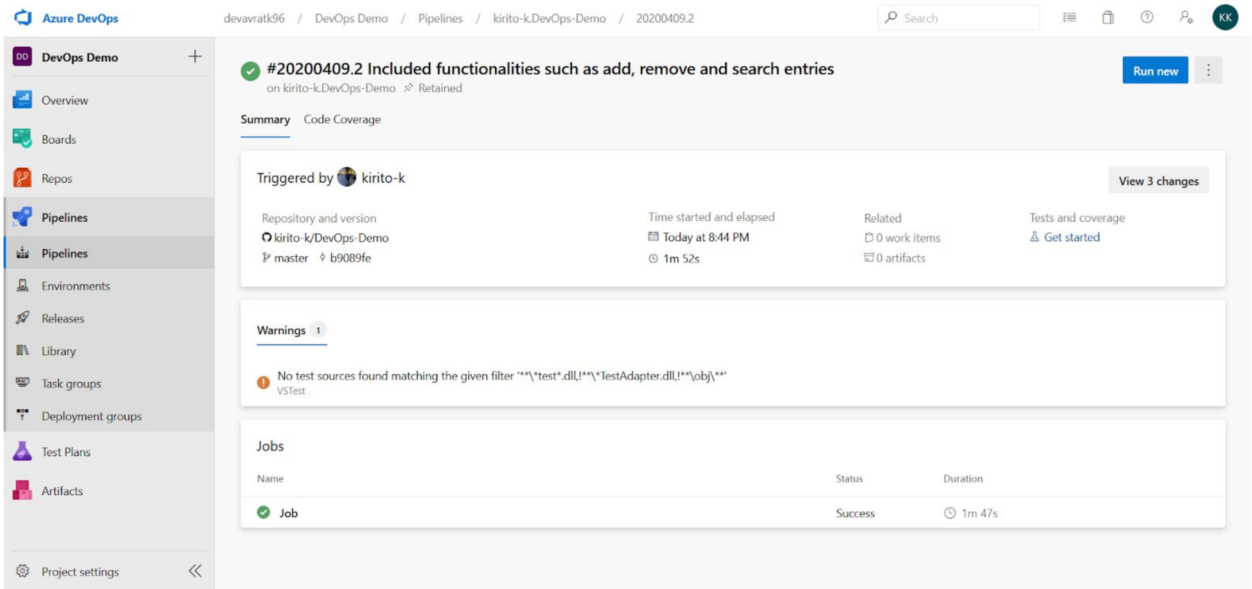
80th percentile

1m 35s Succeeded runs: 1

51.58% of total run time is taken by step - *VSBuild*

View full report

8. We can also see detailed reports of a particular run by selecting it.



Azure DevOps

devavratk96 / DevOps Demo / Pipelines / kirito-k.DevOps-Demo / 20200409.2

Search

KK

DevOps Demo

Overview

Boards

Repos

Pipelines

Pipelines

Environments

Releases

Library

Task groups

Deployment groups

Test Plans

Artifacts

Project settings

#20200409.2 Included functionalities such as add, remove and search entries

on kirito-k.DevOps-Demo Retained

Run new

Summary Code Coverage

Triggered by kirito-k

View 3 changes

Repository and version	Time started and elapsed	Related	Tests and coverage
kirito-k/DevOps-Demo	Today at 8:44 PM	0 work items	Get started
master b9089fe	1m 52s	0 artifacts	

Warnings 1

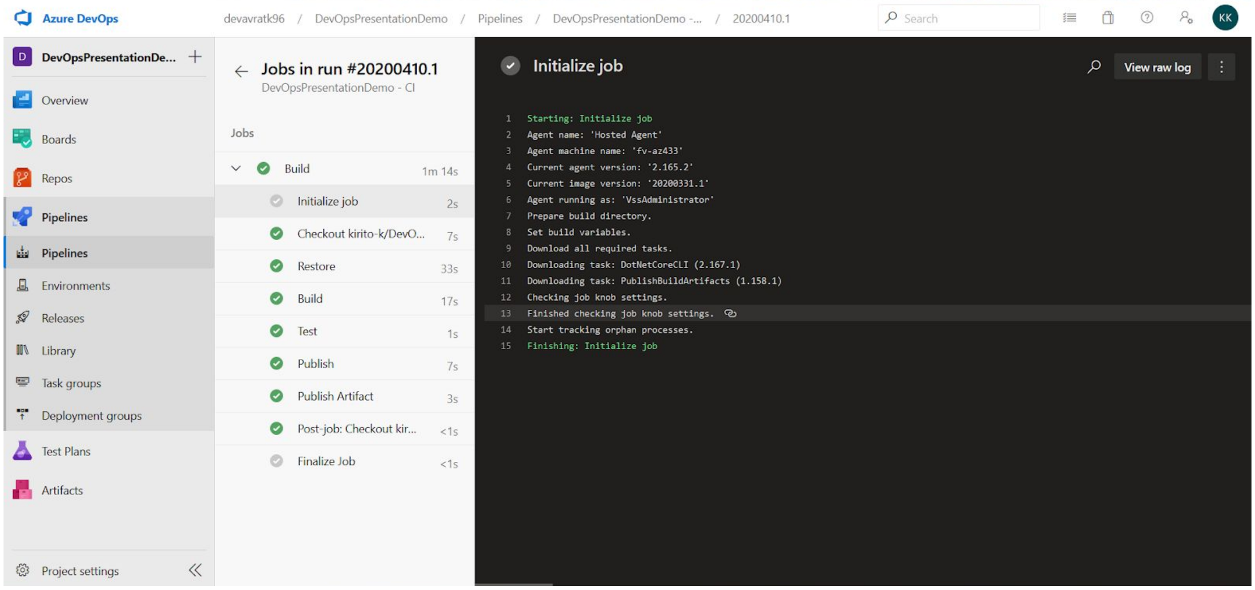
No test sources found matching the given filter '***test*.dll,!***TestAdapter.dll,!**\obj**'

VSTest

Jobs

Name	Status	Duration
Job	Success	1m 47s

9. Click on Job to see the detailed process. We can also add tests for our project to run over.



Jobs in run #20200410.1
DevOpsPresentationDemo - CI

Jobs

Build	1m 14s
Initialize job	2s
Checkout kirito-k/DevO...	7s
Restore	33s
Build	17s
Test	1s
Publish	7s
Publish Artifact	3s
Post-job: Checkout kir...	<1s
Finalize Job	<1s

Initialize job

```
Starting: Initialize job
Agent name: 'Hosted Agent'
Agent machine name: 'fv-az433'
Current agent version: '2.165.2'
Current image version: '20200331.1'
Agent running as: 'VssAdministrator'
Prepare build directory.
Set build variables.
Download all required tasks.
Downloading task: DotNetCoreCLI (2.167.1)
Downloading task: PublishBuildArtifacts (1.158.1)
Checking job knob settings.
Finished checking job knob settings.
Start tracking orphan processes.
Finishing: Initialize job
```

Azure Artifacts

Azure Artifacts is a fully integrated package management for the continuous integration/continuous delivery (CI/CD) pipelines. It creates and shares Maven, npm, NuGet, and Python package feeds from public and private sources with teams of any size. Azure Articants features sharing of code efficiently, managing all package types and add packages to our pipelines. In our example, we can control which version of NuGet packages to use like Microsoft Entity Frameworks.

Azure Boards

Azure Boards is a service for managing the work for your software projects. Teams need tools that flex and grow. Azure Boards does just that, bringing you a rich set of capabilities including native support for Scrum and Kanban, customizable dashboards, and integrated reporting. Boards, in general, is an agile tool to plan, track, and discuss work across your teams.

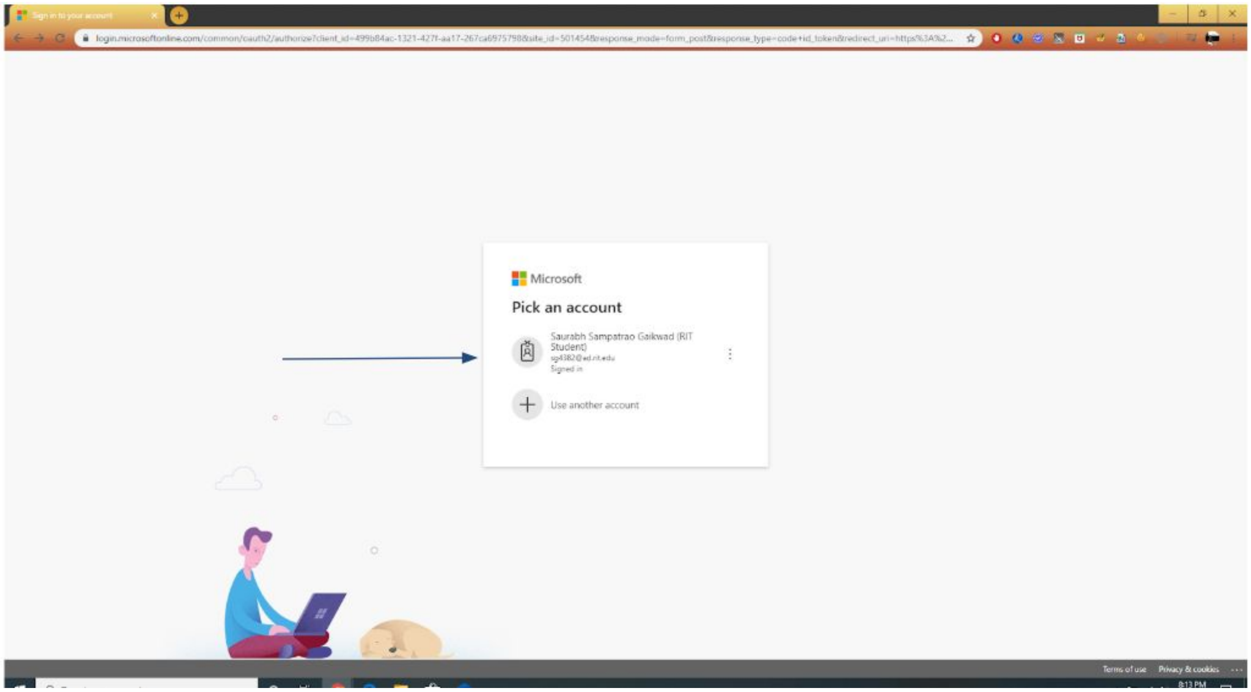
Why Boards?

Anyone who has worked on a software project knows that there are issues to track, manage and prioritize.

Demo: Azure Boards

In Azure boards, we are going to look into how to build a board for your project and backlog, how to run a sprint and track important work.

1. Login to this link


Sign in to your account
login.microsoftonline.com/common/oauth2/authorize?client_id=499b84ac-1321-427f-aa17-267ca6975798&site_id=501454&response_mode=form_post&response_type=code+id_token&redirect_uri=https%3A%2...
Terms of use
Privacy & cookies

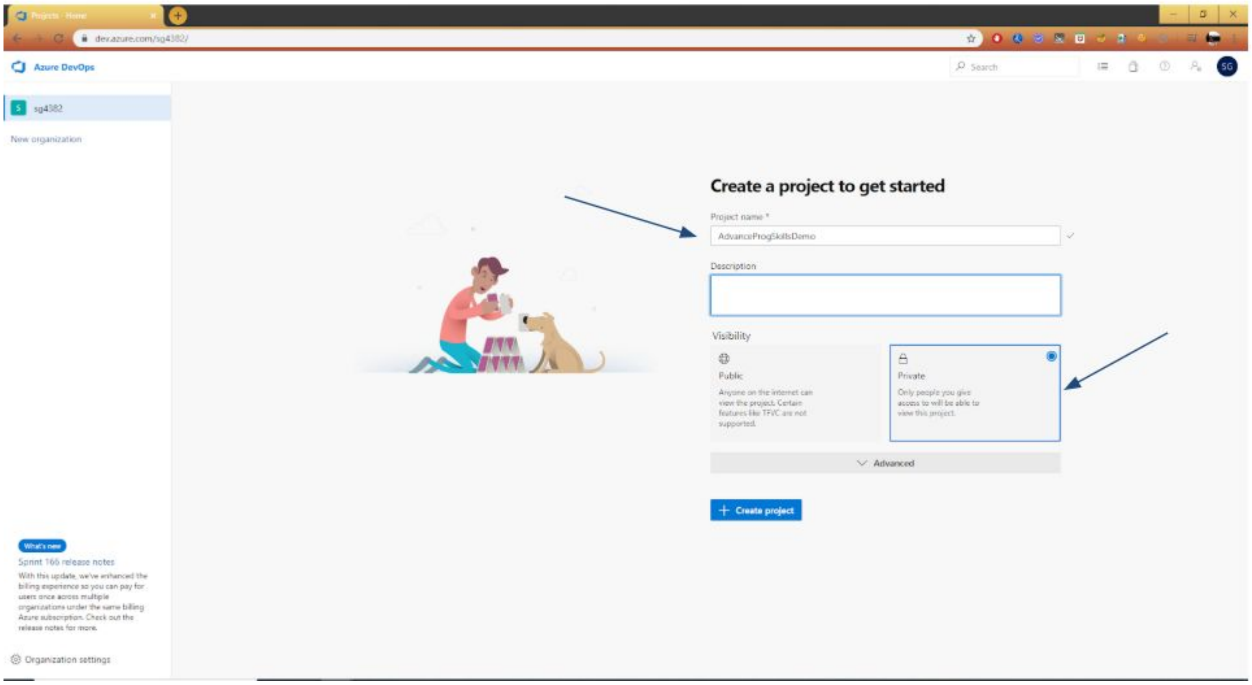
Microsoft

Pick an account

Saurabh Sampatrao Gaikwad (RIT Student)
sg4382@ad.rit.edu
Signed in

Use another account

2. Create a project. Type the project name, enter the description (not mandatory) and you can select public or private visibility. I have selected private because I want only those people to look at the progress to whom I give access.



Sprint 166 release notes

With this update, we've enhanced the billing experience so you can pay for users once across multiple organizations under the same billing Azure subscription. Check out the release notes for more.

Create a project to get started

Project name *: AdvanceProgSkillsDemo

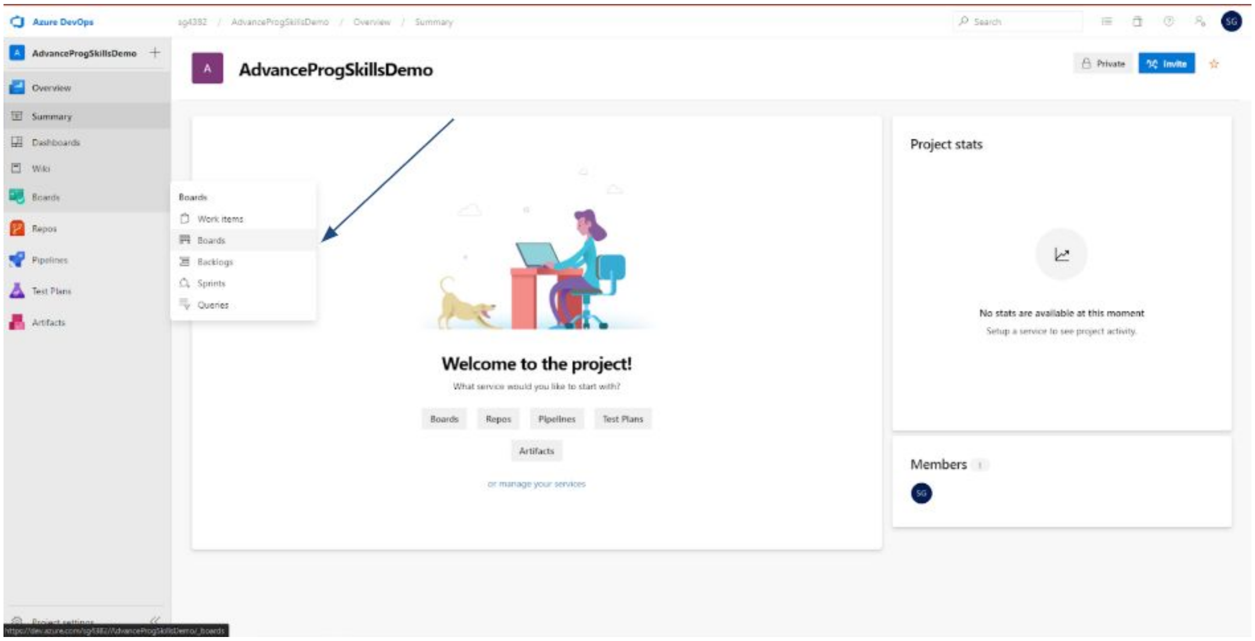
Description:

Visibility

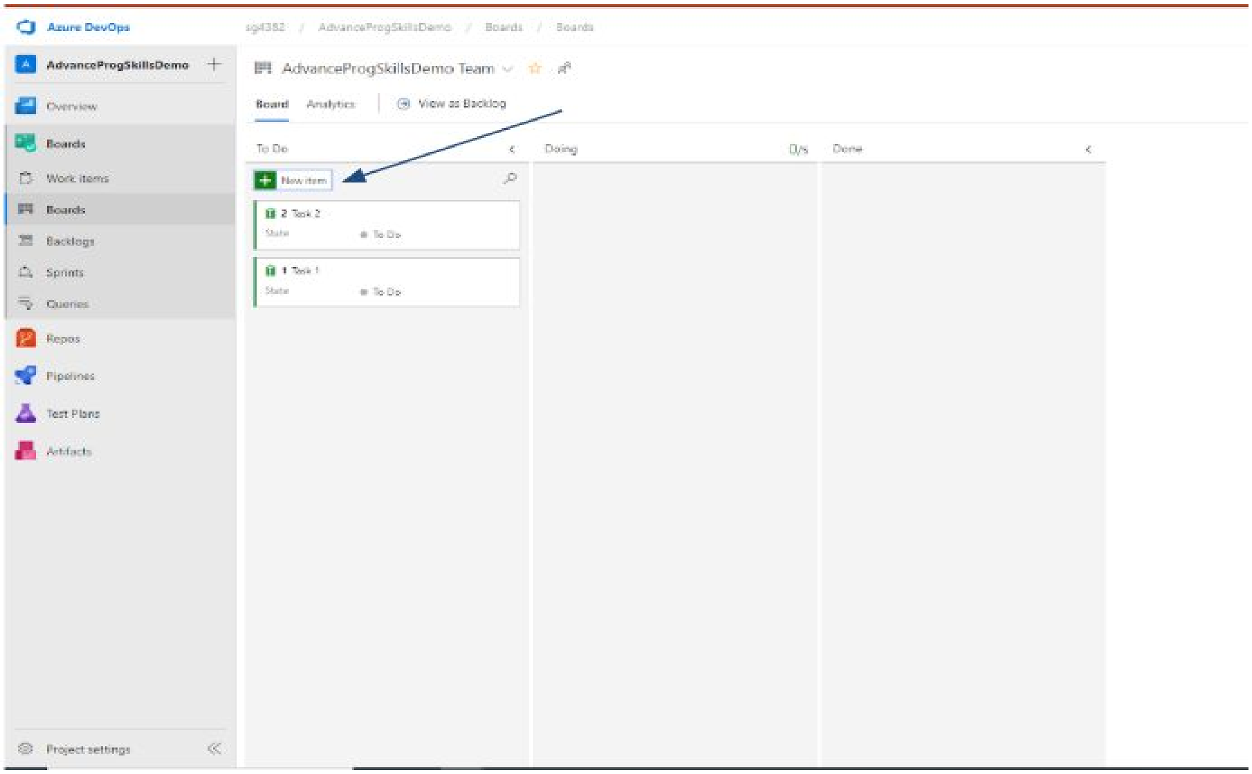
○ Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

● Private
Only people you give access to will be able to view this project.

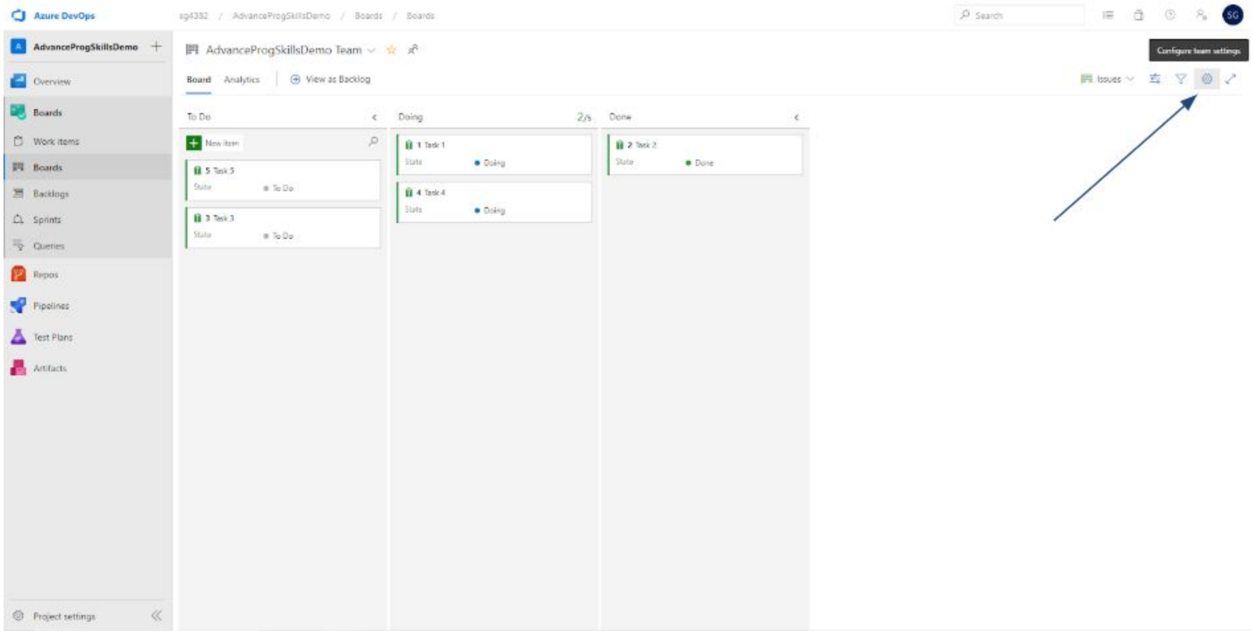
3. We select Boards from Boards option on the left pane.



4. You can see three columns, “To do”, “doing” and “Done”. We can customize the columns if we need to add or delete any columns.



5. To create a task, you have to click on New Item in the “To do” column. You can create as many tasks as you want. You can just drag the task from column to column. Let us say Task 1 and Task 4 are in Doing State (some person is working on the task), Task 5 and Task 3 are still not started by anyone and Task 2 is Done (completed). The board might look something like this.
   Click on Configure Team Settings option to add/edit columns.

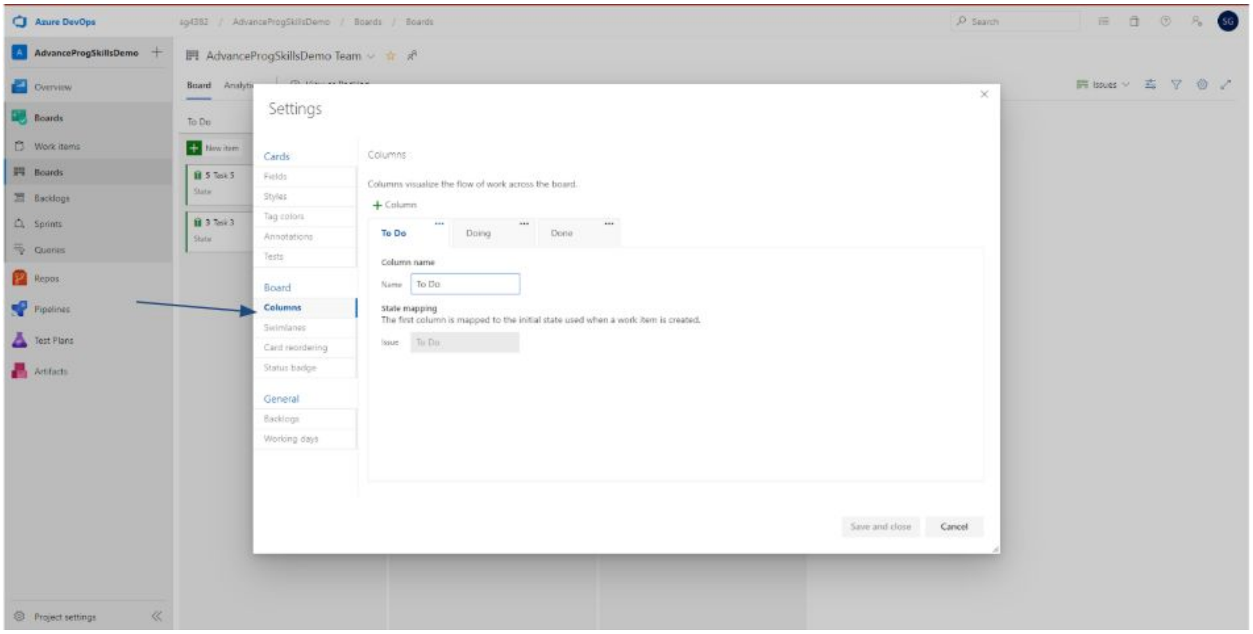


To Do:
- 5 Task 5 — State ● To Do
- 3 Task 3 — State ● To Do

Doing:
- 1 Task 1 — State ● Doing
- 4 Task 4 — State ● Doing

Done:
- 2 Task 2 — State ● Done

6. Clicking on the Setting we can see the following options. There are options of
   Cards -> Fields, Styles, Tag colors, Annotations and Tests.
   Board -> Columns, Swimlanes, Card reordering, Status Badge
   General -> Backlogs, Working days.



Columns

Columns visualize the flow of work across the board.

Column name

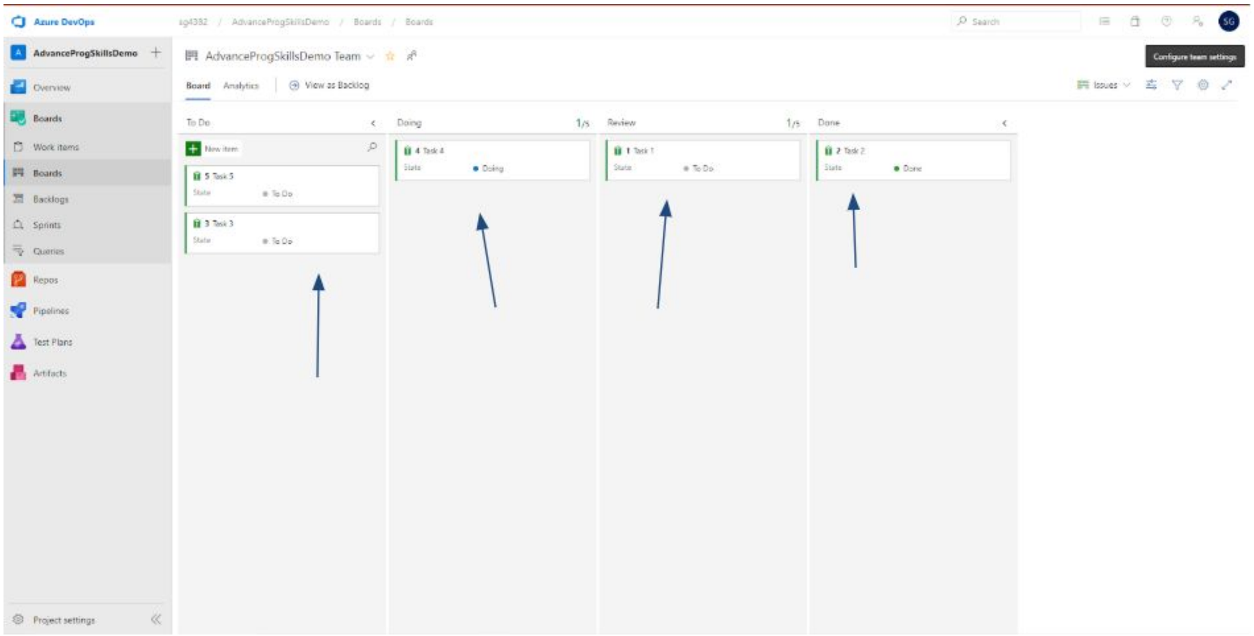
Name: To Do

State mapping

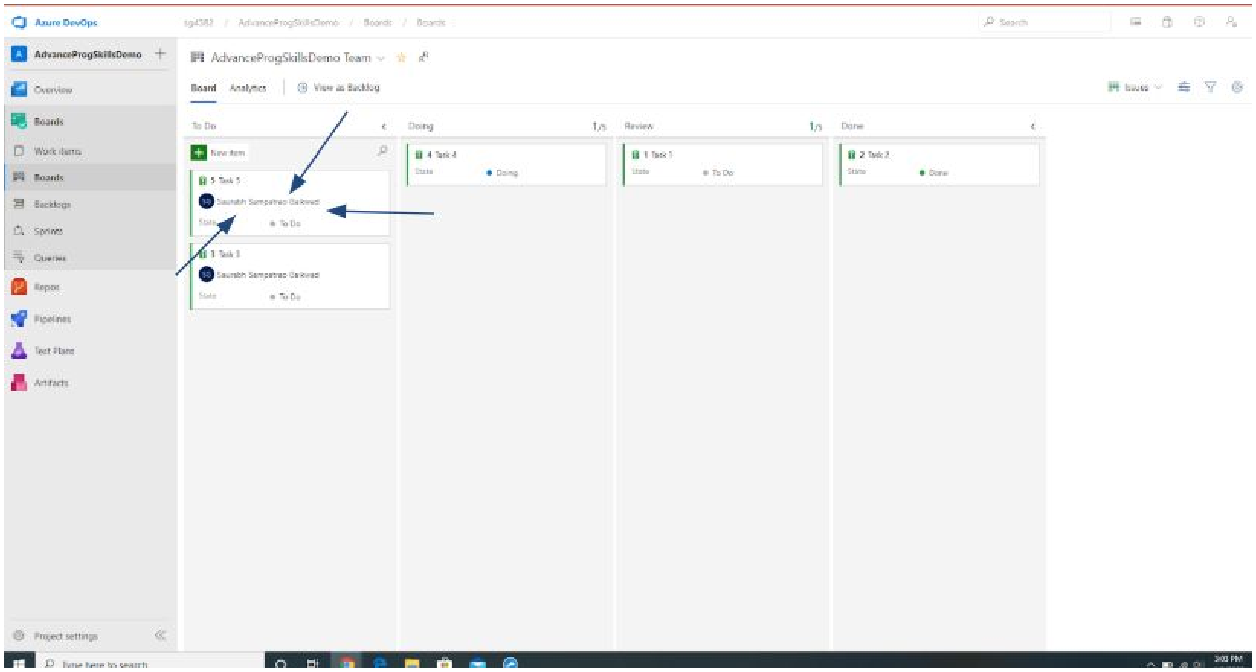
The first column is mapped to the initial state used when a work item is created.

Issue: To Do

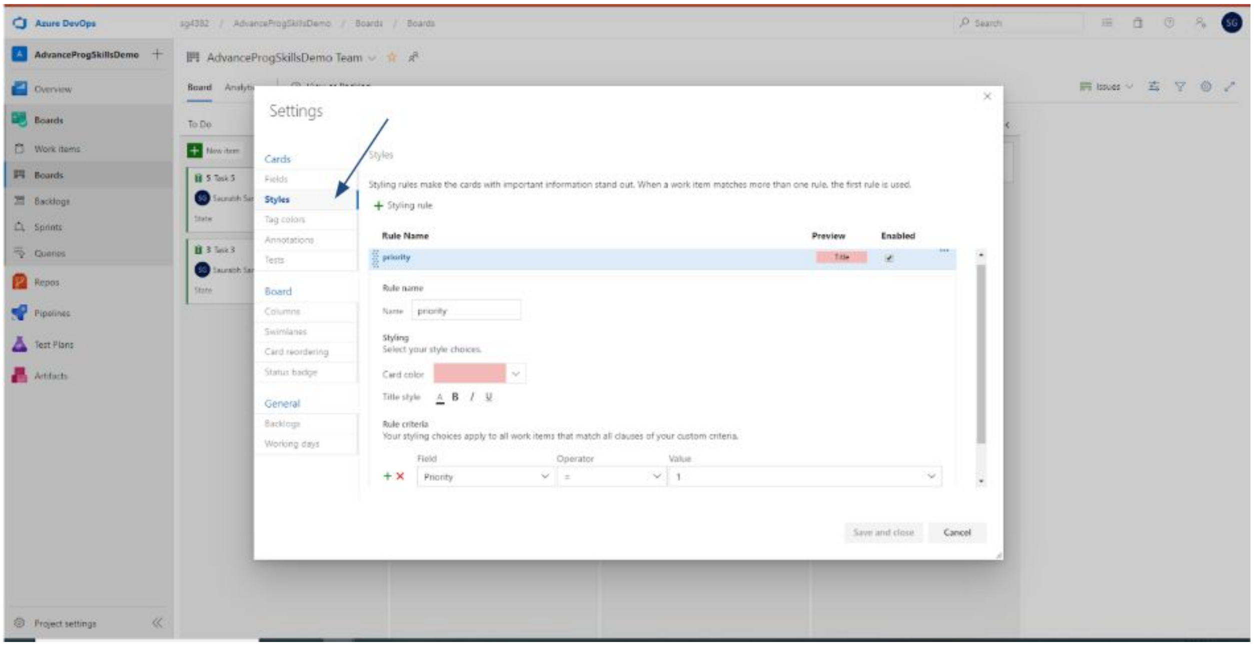
7. This is how it looks when we add a new column in the board of our project.



8. Here you can see that in Column To Do I have assigned the tasks to myself. There's just myself in the team. You can give access and add others in the project and assign them tasks.



9. You can style the board by coloring the tasks as per the priority, First we create a rule in Setting -> Styles like this. The rule created is if Priority = 1, change the color of the task to light red.



Settings

Styles

Styling rules make the cards with important information stand out. When a work item matches more than one rule, the first rule is used.

Rule Name	Preview	Enabled
priority	Title	✓

Rule name

Name: priority

Styling

Select your style choices.

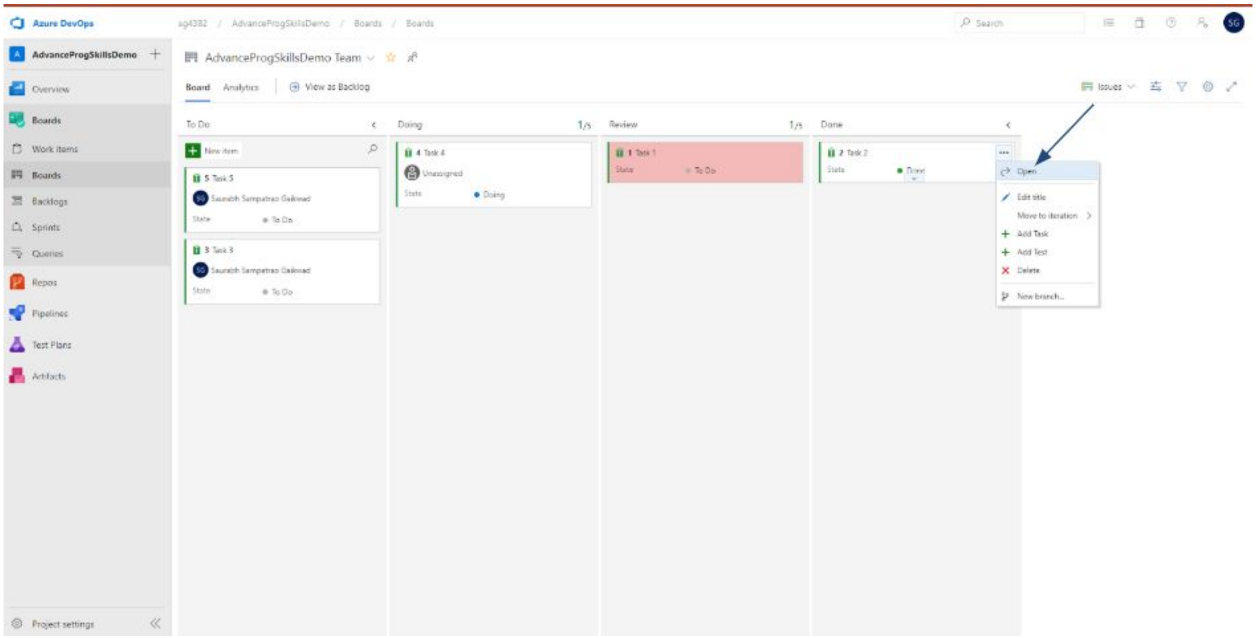
Card color

Rule criteria

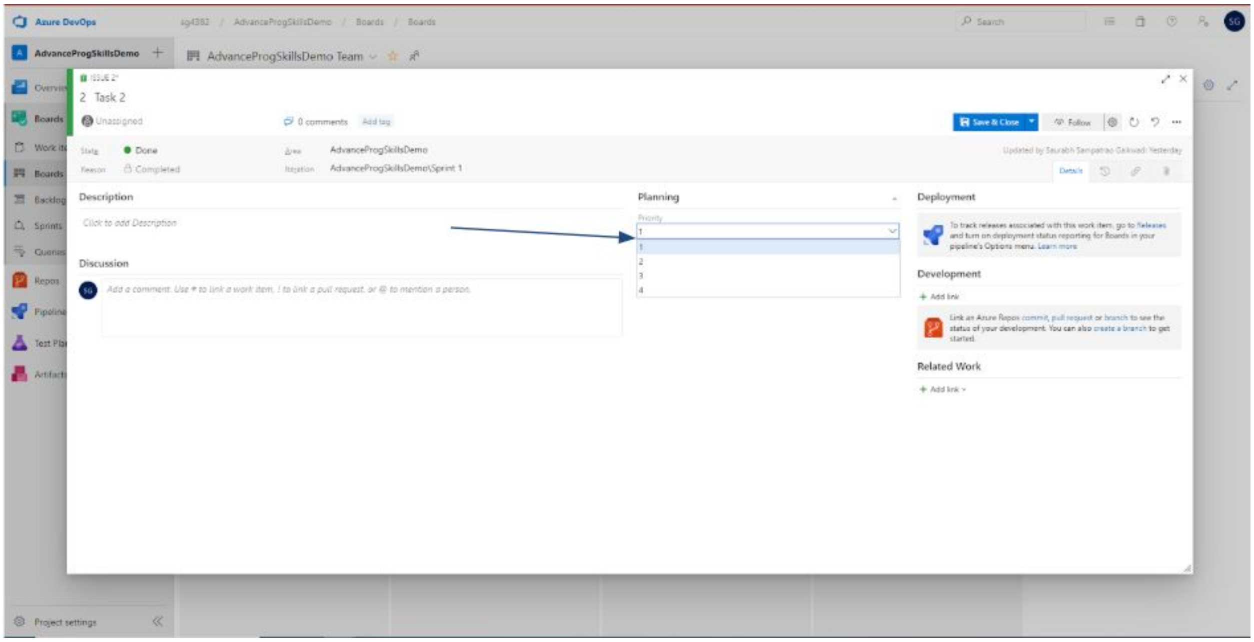
Your styling choices apply to all work items that match all clauses of your custom criteria.

Field	Operator	Value
Priority	=	1

10. We open a task by clicking the three dots.



11. We set the priority as 1.



ISSUE 2

2 Task 2

Unassigned

0 comments

State: Done

Reason: Completed

Area: AdvanceProgSkillsDemo

Iteration: AdvanceProgSkillsDemo\Sprint 1

Updated by Saurabh Sampatrao Gaikwad: Yesterday

Description

Click to add Description

Discussion

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

Planning

Priority: 1

Deployment

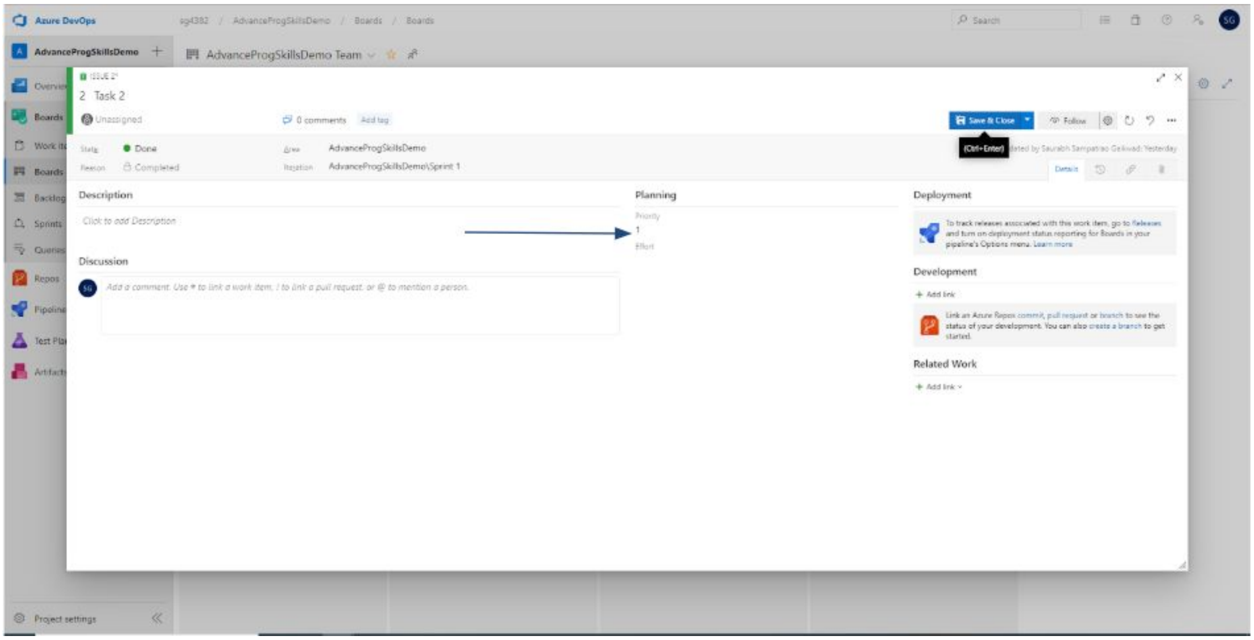
To track releases associated with this work item, go to Releases and turn on deployment status reporting for Boards in your pipeline's Options menu. Learn more

Development

Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.

Related Work

12. We save and close the task.



ISSUE 2

2 Task 2

Unassigned

0 comments

State: Done

Reason: Completed

Area: AdvanceProgSkillsDemo

Iteration: AdvanceProgSkillsDemo\Sprint 1

Updated by Saurabh Sampatrao Gaikwad: Yesterday

Description

Click to add Description

Discussion

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

Planning

Priority: 1

Effort

Deployment

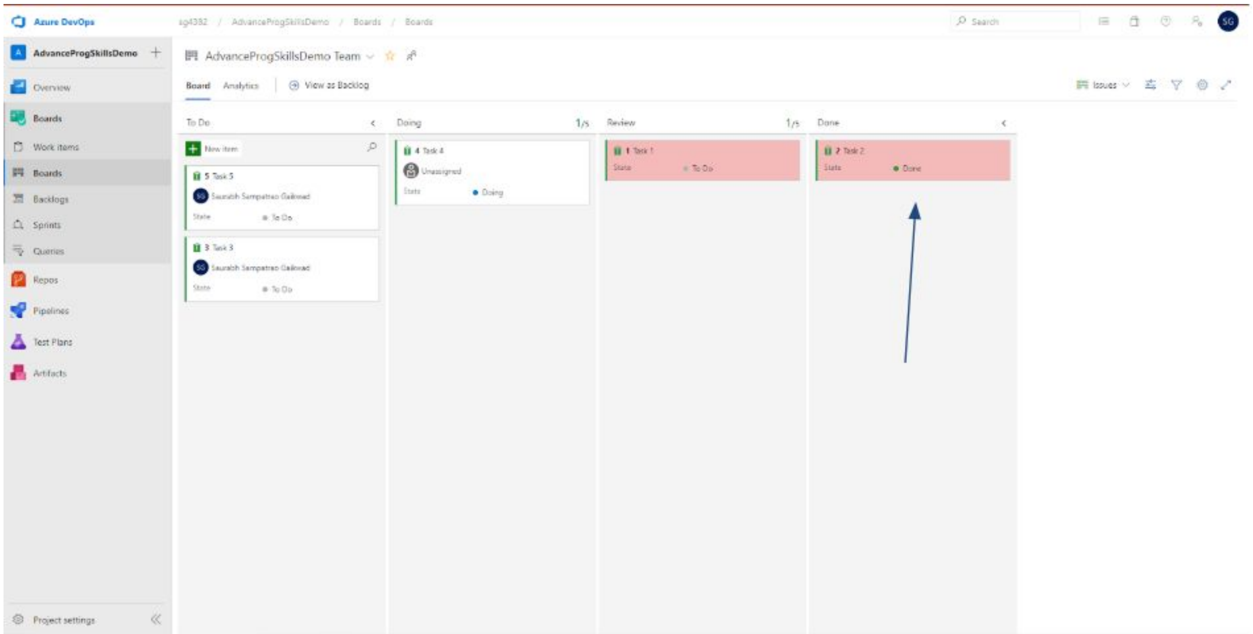
To track releases associated with this work item, go to Releases and turn on deployment status reporting for Boards in your pipeline's Options menu. Learn more

Development

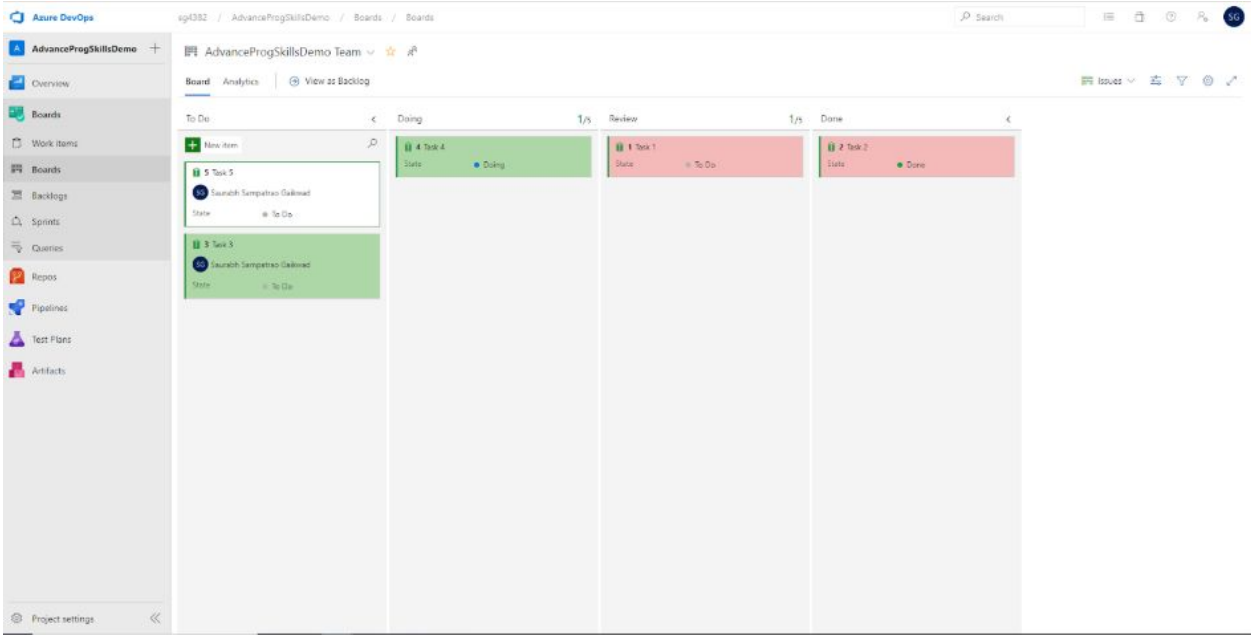
Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.

Related Work

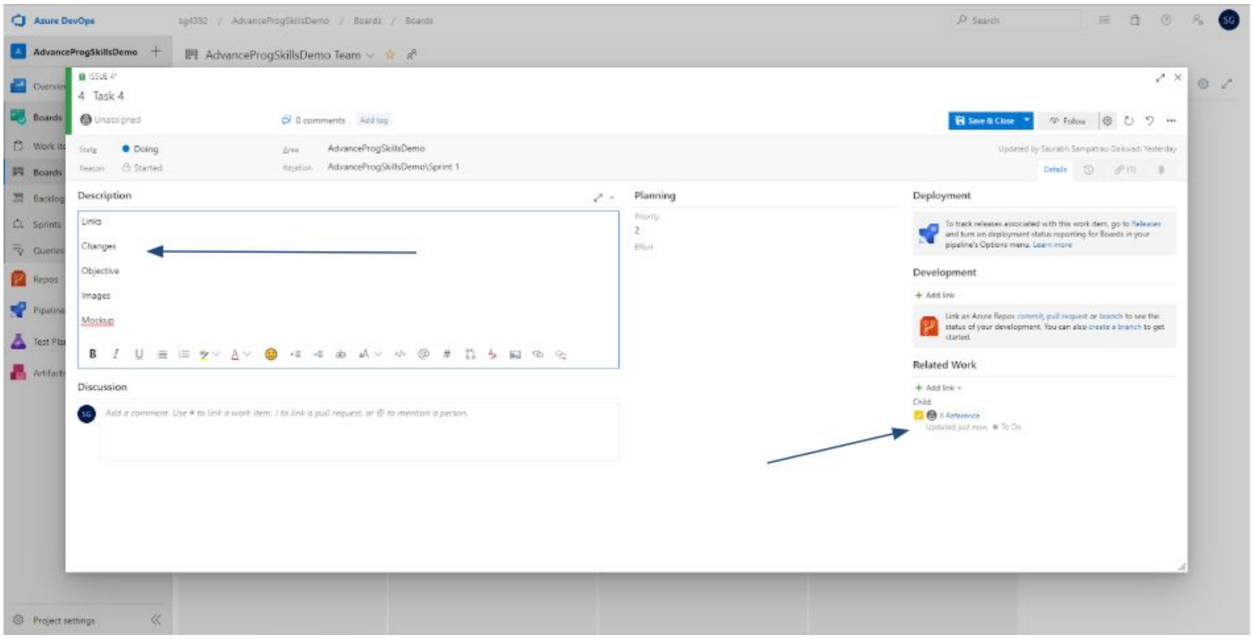
13. We can see the changed color of task 2.



14. I have already changed the color of task 1. Here's how it looks.



15. We can add a description to the task, on the left highlighted box. We can also add links to other related work on the right side bottom. I have already added one link to show how it looks. You can add more to Planning, as we create new rules.



4 Task 4

Unassigned

0 comments Add tag

State: Doing

Reason: Started

Area: AdvanceProgSkillsDemo

Iteration: AdvanceProgSkillsDemo\Sprint 1

Updated by Saurabh Sampatrao Gaikwad: Yesterday

Description:

Links

Changes

Objective

Images

Mockup

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

Planning

Priority: 2

Effort

Deployment: To track releases associated with this work item, go to Releases and turn on deployment status reporting for Boards in your pipeline's Options menu. Learn more

Development: Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.

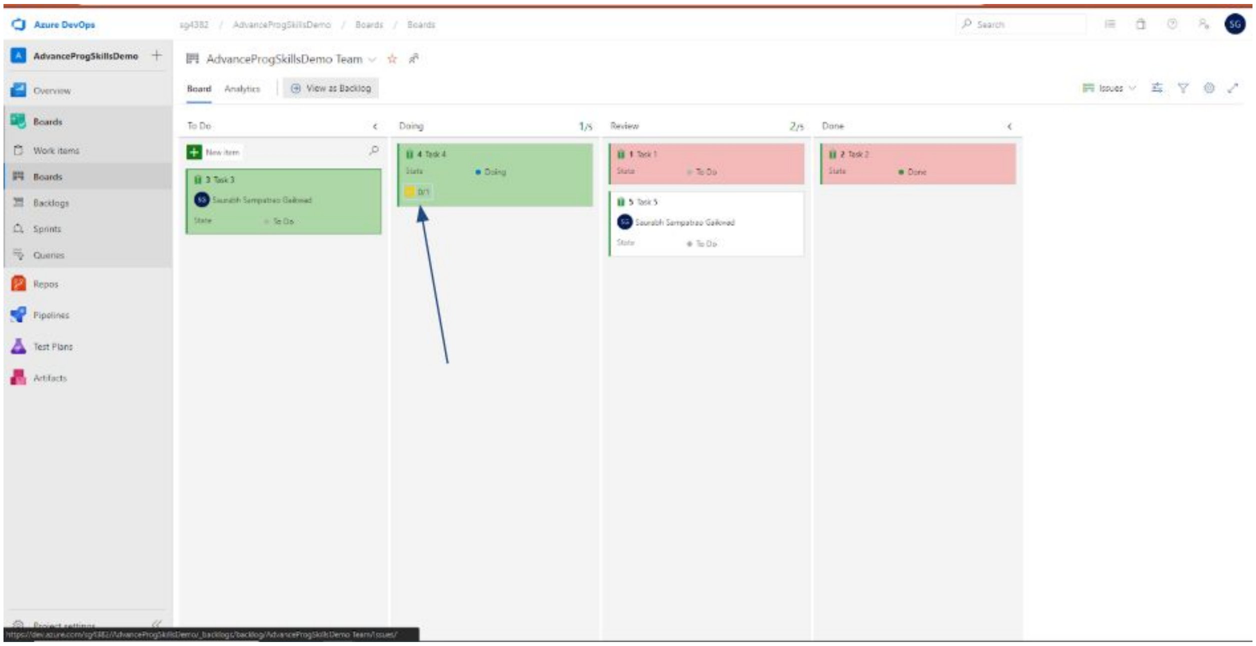
Related Work

Child

6 Reference

Updated just now, To Do

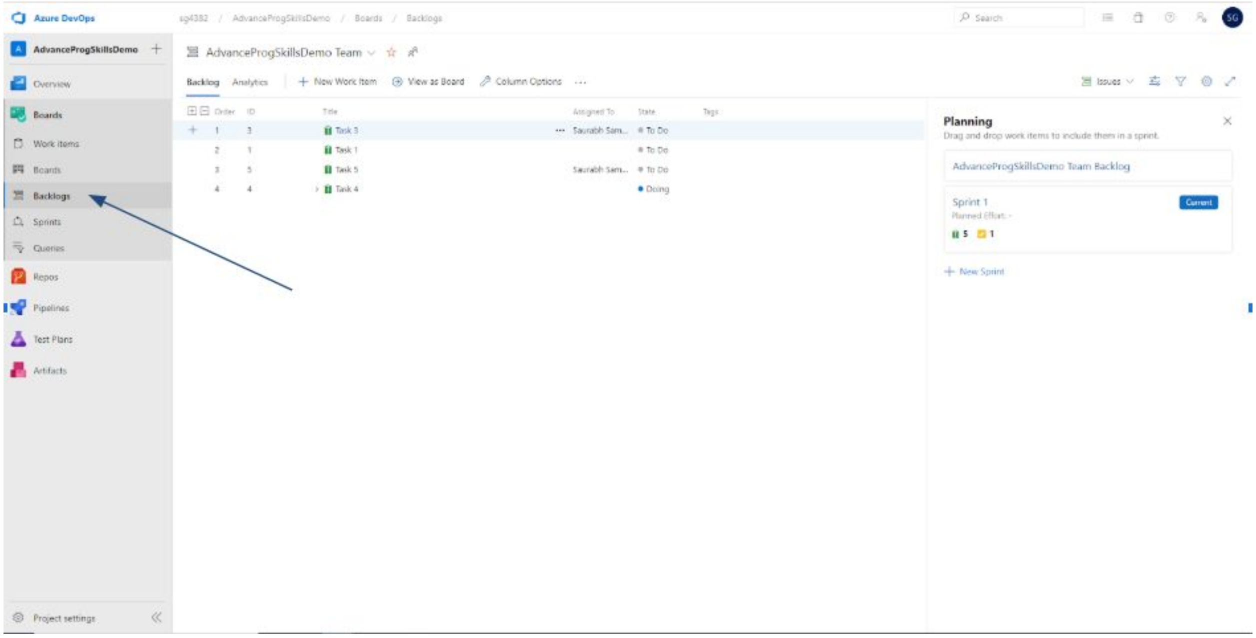
16. Let’s see how to create a sprint and view the backlog. Click on View as Backlog.



To Do	Doing 1/5	Review 2/5	Done
3 Task 3 Saurabh Sampatrao Gaikwad State: To Do	4 Task 4 State: Doing 0/1	1 Task 1 State: To Do	2 Task 2 State: Done
		5 Task 5 Saurabh Sampatrao Gaikwad State: To Do	

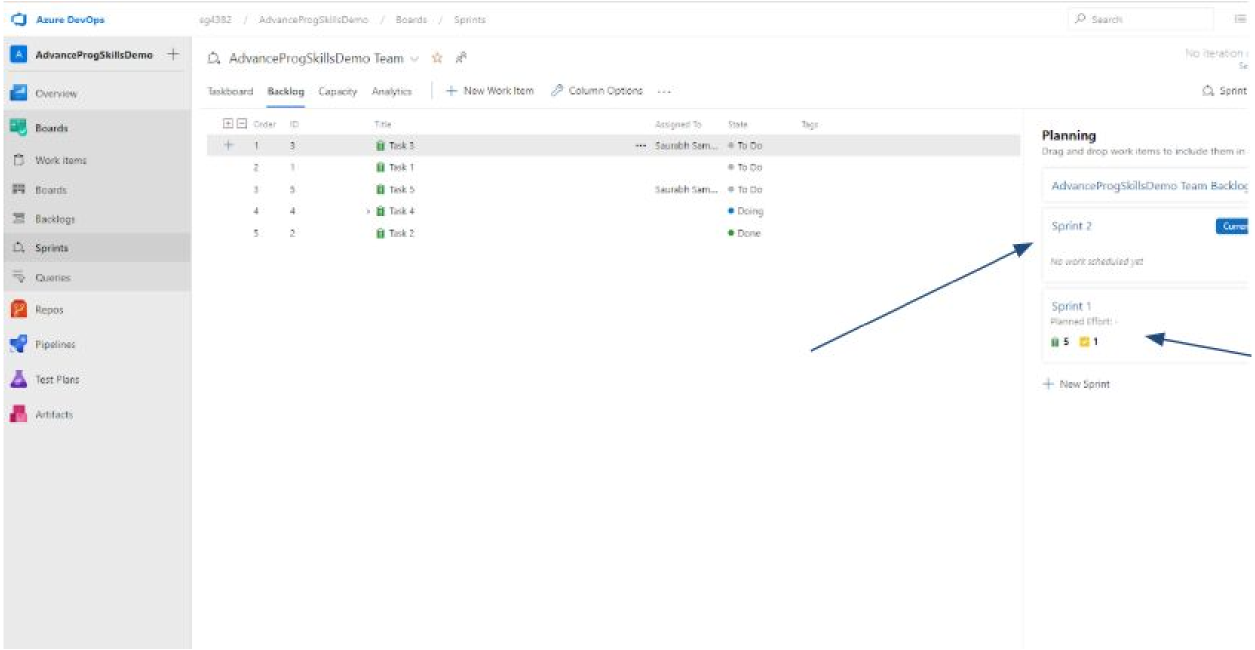
https://dev.azure.com/sg4382/AdvanceProgSkillsDemo/_backlogs/backlog/AdvanceProgSkillsDemo Team/Issues/

17. Click on New Sprint.



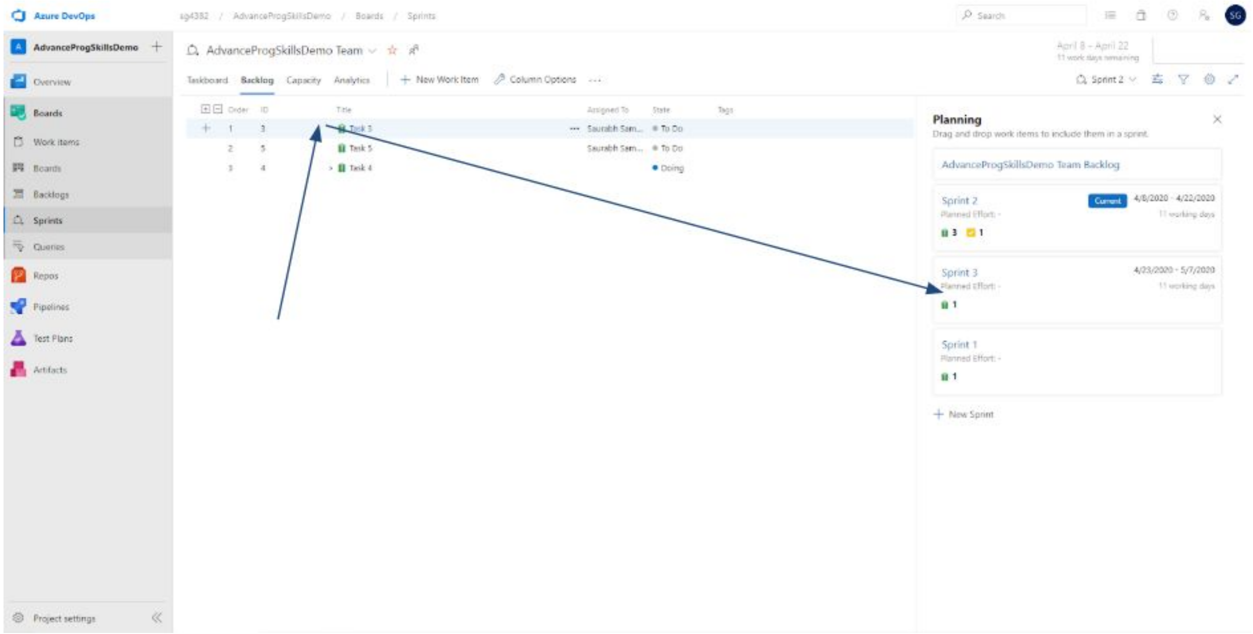
Order	ID	Title	Assigned To	State	Tags
1	3	Task 3	Saurabh Sam...	To Do	
2	1	Task 1		To Do	
3	5	Task 5	Saurabh Sam...	To Do	
4	4	Task 4		Doing	

18. We can see Sprint 2 created on the right pane. Suppose there is a task we cannot complete in this sprint we can drag and move the task to another sprint.



Order	ID	Title	Assigned To	State	Tags
1	3	Task 3	Saurabh Sam...	To Do	
2	1	Task 1		To Do	
3	5	Task 5	Saurabh Sam...	To Do	
4	4	Task 4		Doing	
5	2	Task 2		Done	

19. This will look like this, we have dragged To Do and Doing tasks to Sprint 2. This is a backlog view.



Order	ID	Title	Assigned To	State	Tags
1	3	Task 3	Saurabh Sam...	To Do	
2	5	Task 5	Saurabh Sam...	To Do	
3	4	Task 4		Doing	

Planning
Drag and drop work items to include them in a sprint.

AdvanceProgSkillsDemo Team Backlog

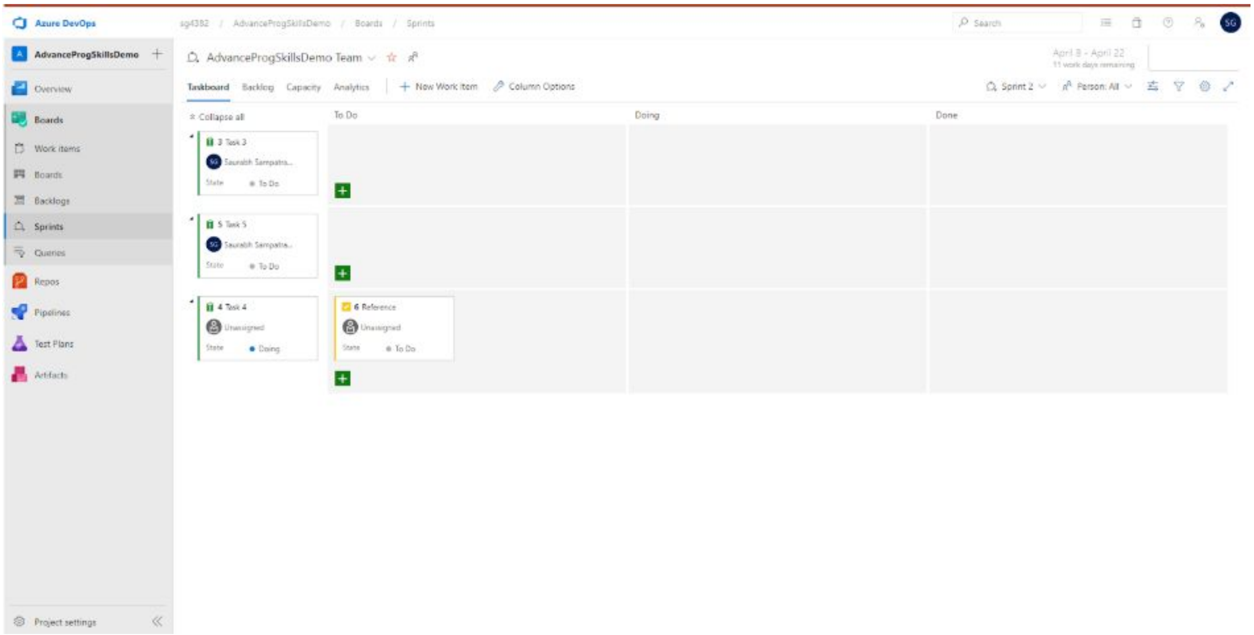
Sprint 2 Current 4/8/2020 - 4/22/2020
Planned Effort: - 11 working days
3 1

Sprint 3 4/23/2020 - 5/7/2020
Planned Effort: - 11 working days
1

Sprint 1
Planned Effort: -
1

New Sprint

20. There is a Task Board View to the left of the Backlog view.



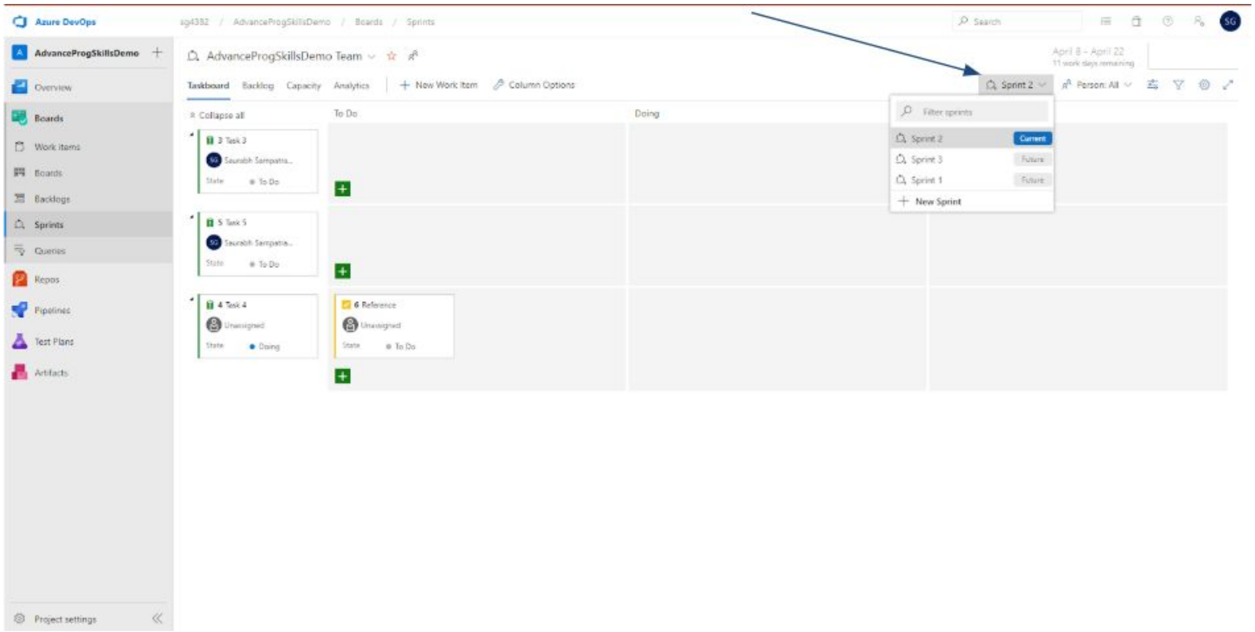
3 Task 3
Saurabh Sampatra...
State To Do

5 Task 5
Saurabh Sampatra...
State To Do

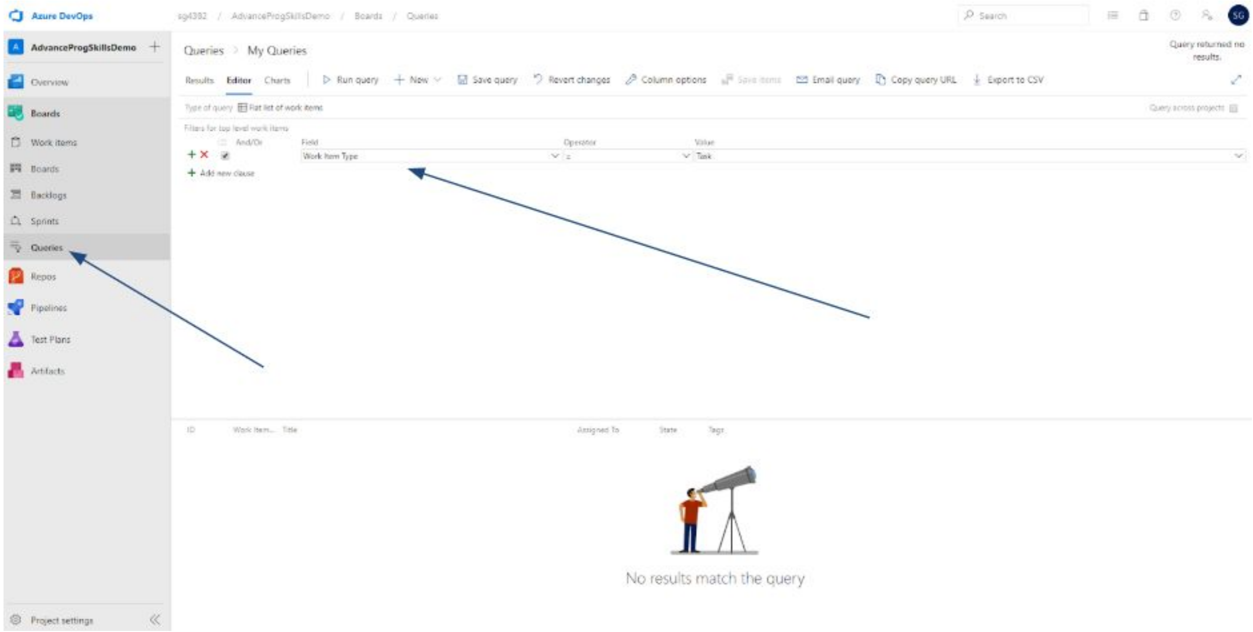
4 Task 4
Unassigned
State Doing

6 Reference
Unassigned
State To Do

21. You can change the sprint and view it’s tasks by clicking on the Sprint option.



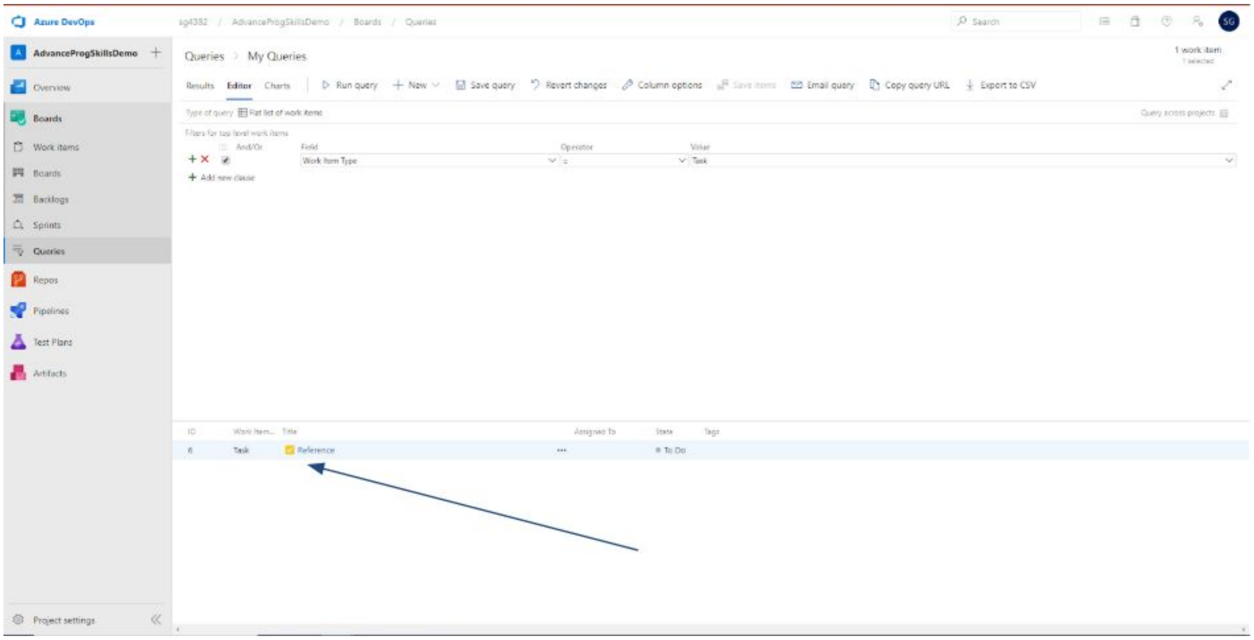
22. To keep track Azure Boards have a Query option. We can create a query and all the Issues, tasks, bugs are seen in the result.
Example: We created a query to show the work type item field contains the “Task” keyword.



Type of query: Flat list of work items

And/Or	Field	Operator	Value
☑	Work Item Type	=	Task

23. When we Run the query we can see the results below. The task with ID 6 is displayed. This is how we can keep a track of the project in the longer run.



Type of query: Flat list of work items

Filters for top level work items

And/Or	Field	Operator	Value
☑	Work Item Type	=	Task

+ Add new clause

ID	Work Item...	Title	Assigned To	State	Tags
6	Task	Reference		To Do	

AZURE Repos

If you are writing code, then you need a place to store and manage that code with a reliable version control system like Git. Azure Repos provides one of the best in class Git solutions. With Azure Repos you get free private and public repos, social code reviews, and more. In this video we will walk you on how to get started with Git in Azure Repos and how your team can use pull requests to work together to collaborate on code.

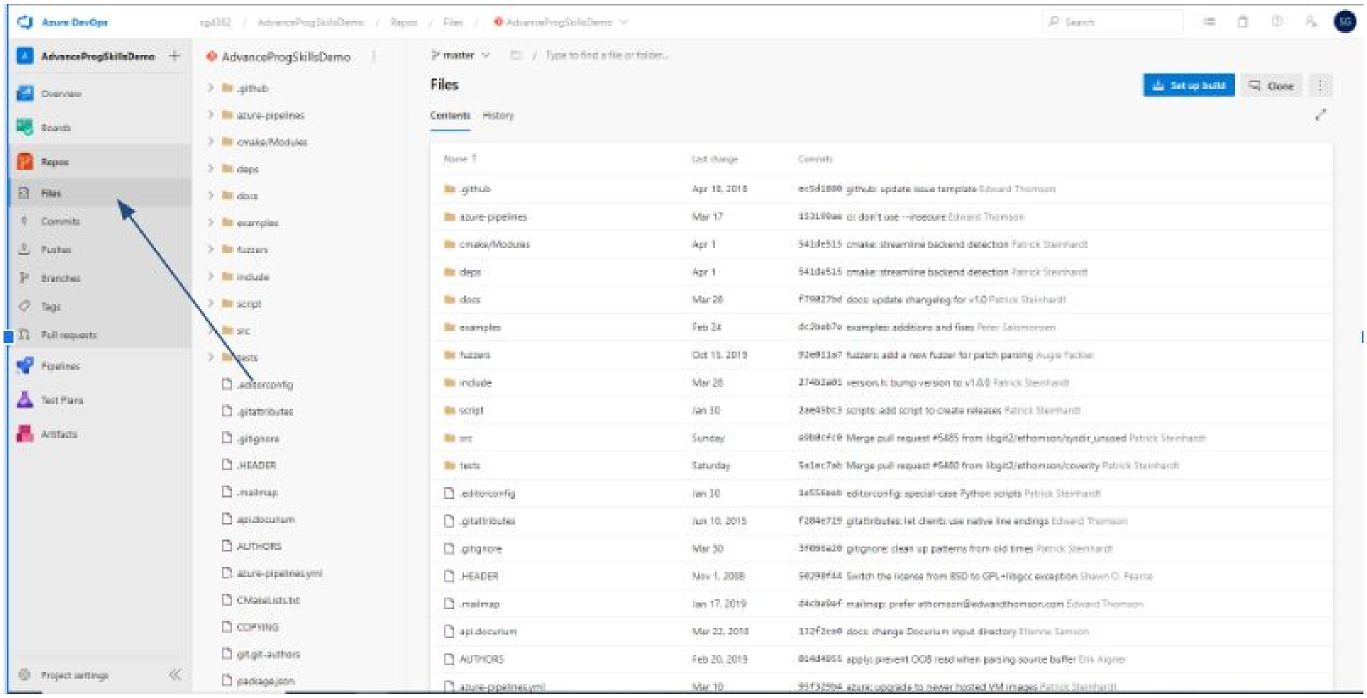
Why Repos?

It helps in collaborating to build better code. Performs more effectively in Git code reviews with threaded discussion and continuous integration. We can automate with built-in CI/CD. Protects your code quality with branch policies.

Demo: Azure Repos

I have pulled a project from gitlab to show a demo. The URL is https://github.com/libgit2/libgit2.

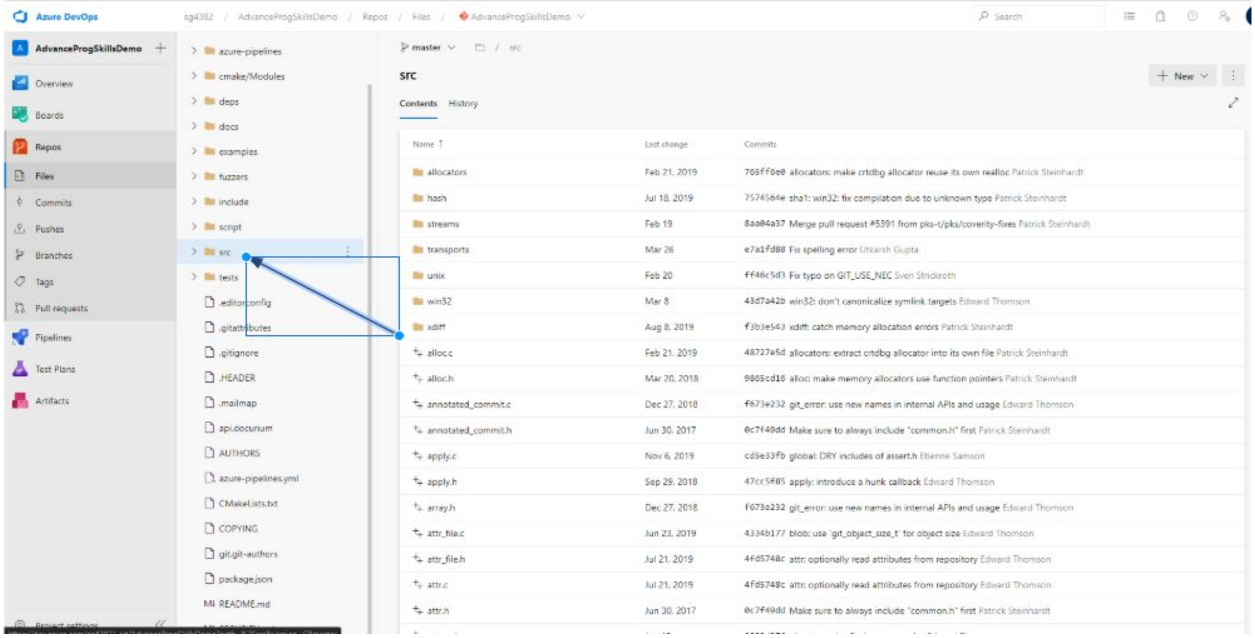
1. The repository looks like this after the pull request. Here you can browse the repository structure. You can click on any folder to navigate through the folders.



Files

Name ↑	Last change	Commits
.github	Apr 18, 2018	ec5d1800 github: update issue template Edward Thomson
azure-pipelines	Mar 17	153180ae ci: don't use --insecure Edward Thomson
cmake/Modules	Apr 1	541de515 cmake: streamline backend detection Patrick Steinhardt
deps	Apr 1	541de515 cmake: streamline backend detection Patrick Steinhardt
docs	Mar 28	f79027bd docs: update changelog for v1.0 Patrick Steinhardt
examples	Feb 24	dc2bab7e examples: additions and fixes Peter Salomonsen
fuzzers	Oct 15, 2019	92e011a7 fuzzers: add a new fuzzer for patch parsing Augie Fackler
include	Mar 28	37402a01 version.h: bump version to v1.0.0 Patrick Steinhardt
script	Jan 30	2ae45bc3 scripts: add script to create releases Patrick Steinhardt
src	Sunday	a908cfc8 Merge pull request #5485 from libgit2/ethomson/sysdir_unused Patrick Steinhardt
tests	Saturday	5a1ec7eb Merge pull request #5480 from libgit2/ethomson/coverity Patrick Steinhardt
.editorconfig	Jan 30	1a556eeb editorconfig: special-case Python scripts Patrick Steinhardt
.gitattributes	Jun 10, 2015	f284e719 gitattributes: let clients use native line endings Edward Thomson
.gitignore	Mar 30	39066a28 gitignore: clean up patterns from old times Patrick Steinhardt
.HEADER	Nov 1, 2008	58298f44 Switch the license from BSD to GPL+libgcc exception Shawn O. Pearce
.mailmap	Jan 17, 2019	d4cba0ef mailmap: prefer ethomson@edwardthomson.com Edward Thomson
api.docurium	Mar 22, 2018	132f2ee0 docs: change Docurium input directory Etienne Samson
AUTHORS	Feb 20, 2019	804d4051 apply: prevent OOB read when parsing source buffer Erik Aigner
azure-pipelines.yml	Mar 10	95f32504 azure: upgrade to newer hosted VM images Patrick Steinhardt

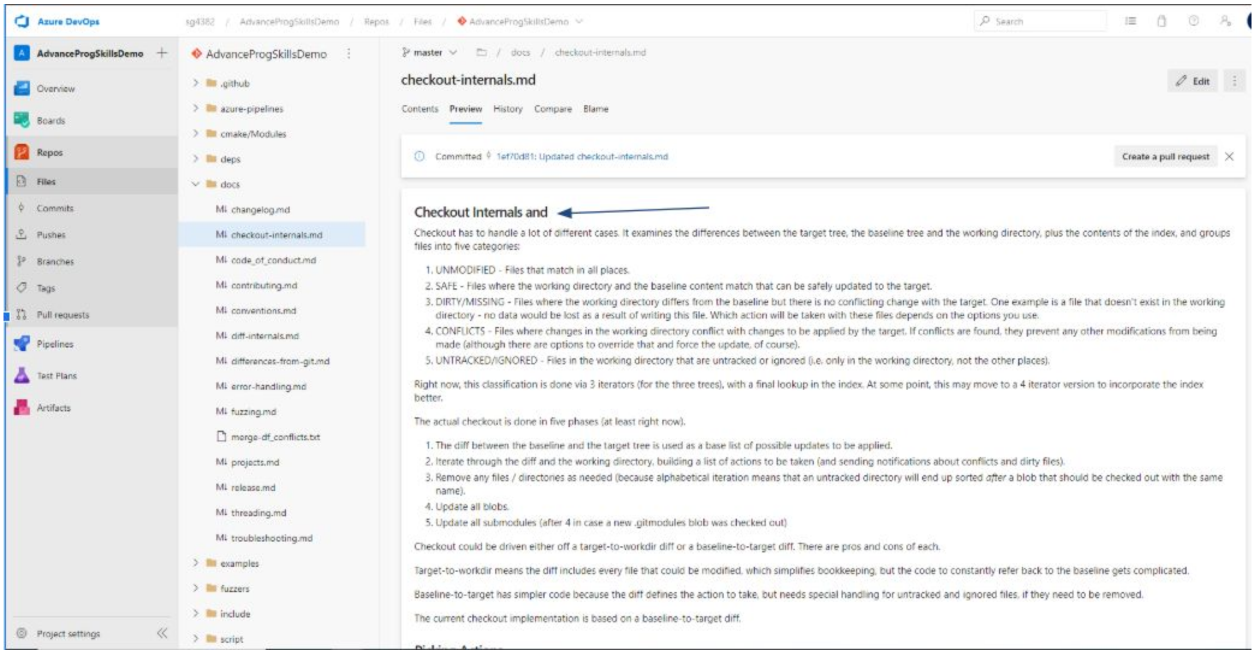
2. If you click on the src folder. You can see the contents of the folder below. We can click on any file and see the contents of the file. The extension of the file does not matter.



src

Name ↑	Last change	Commits
allocators	Feb 21, 2019	765ff6e0 allocators: make crtdbg allocator reuse its own realloc Patrick Steinhardt
hash	Jul 18, 2019	7574564e sha1: win32: fix compilation due to unknown type Patrick Steinhardt
streams	Feb 19	8aa04a37 Merge pull request #5391 from pks-t/pks/coverity-fixes Patrick Steinhardt
transports	Mar 26	e7a1fd88 Fix spelling error Utkarsh Gupta
unix	Feb 20	ff46c5d3 Fix typo on GIT_USE_NEC Sven Strickroth
win32	Mar 8	43d7a42b win32: don't canonicalize symlink targets Edward Thomson
xdiff	Aug 8, 2019	f3b3e543 xdiff: catch memory allocation errors Patrick Steinhardt
alloc.c	Feb 21, 2019	48727e5d allocators: extract crtdbg allocator into its own file Patrick Steinhardt
alloc.h	Mar 20, 2018	9865cd16 alloc: make memory allocators use function pointers Patrick Steinhardt
annotated_commit.c	Dec 27, 2018	f673e232 git_error: use new names in internal APIs and usage Edward Thomson
annotated_commit.h	Jun 30, 2017	0c7f49dd Make sure to always include "common.h" first Patrick Steinhardt
apply.c	Nov 6, 2019	cd5e33fb global: DRY includes of assert.h Etienne Samson
apply.h	Sep 29, 2018	47cc5f85 apply: introduce a hunk callback Edward Thomson
array.h	Dec 27, 2018	f673e232 git_error: use new names in internal APIs and usage Edward Thomson
attr_file.c	Jun 23, 2019	4334b177 blob: use `git_object_size_t` for object size Edward Thomson
attr_file.h	Jul 21, 2019	4fd5748c attr: optionally read attributes from repository Edward Thomson
attr.c	Jul 21, 2019	4fd5748c attr: optionally read attributes from repository Edward Thomson
attr.h	Jun 30, 2017	0c7f49dd Make sure to always include "common.h" first Patrick Steinhardt

3. Let’s see how to make simple changes (typos) in Repos. I have a file which has a typo, you can see that in the next image. I want to remove the word “and” from the first line.



Checkout Internals and

Checkout has to handle a lot of different cases. It examines the differences between the target tree, the baseline tree and the working directory, plus the contents of the index, and groups files into five categories:

1. UNMODIFIED - Files that match in all places.
2. SAFE - Files where the working directory and the baseline content match that can be safely updated to the target.
3. DIRTY/MISSING - Files where the working directory differs from the baseline but there is no conflicting change with the target. One example is a file that doesn't exist in the working directory - no data would be lost as a result of writing this file. Which action will be taken with these files depends on the options you use.
4. CONFLICTS - Files where changes in the working directory conflict with changes to be applied by the target. If conflicts are found, they prevent any other modifications from being made (although there are options to override that and force the update, of course).
5. UNTRACKED/IGNORED - Files in the working directory that are untracked or ignored (i.e. only in the working directory, not the other places).

Right now, this classification is done via 3 iterators (for the three trees), with a final lookup in the index. At some point, this may move to a 4 iterator version to incorporate the index better.

The actual checkout is done in five phases (at least right now).

1. The diff between the baseline and the target tree is used as a base list of possible updates to be applied.
2. Iterate through the diff and the working directory, building a list of actions to be taken (and sending notifications about conflicts and dirty files).
3. Remove any files / directories as needed (because alphabetical iteration means that an untracked directory will end up sorted *after* a blob that should be checked out with the same name).
4. Update all blobs.
5. Update all submodules (after 4 in case a new .gitmodules blob was checked out)

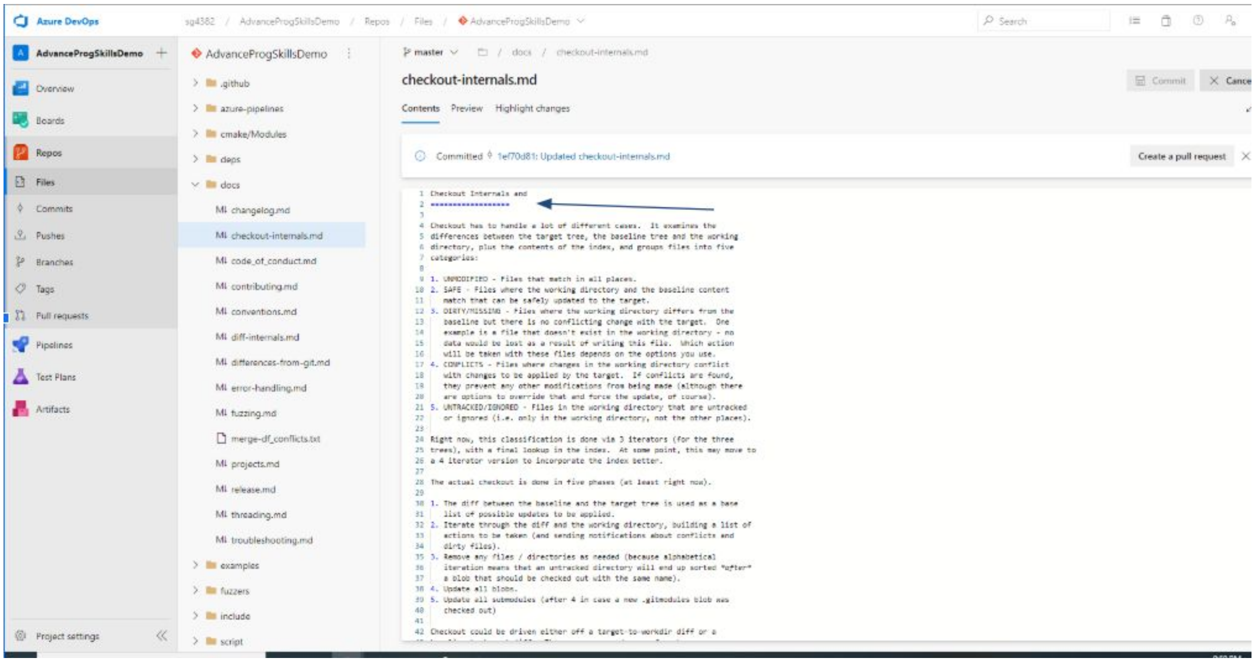
Checkout could be driven either off a target-to-workdir diff or a baseline-to-target diff. There are pros and cons of each.

Target-to-workdir means the diff includes every file that could be modified, which simplifies bookkeeping, but the code to constantly refer back to the baseline gets complicated.

Baseline-to-target has simpler code because the diff defines the action to take, but needs special handling for untracked and ignored files, if they need to be removed.

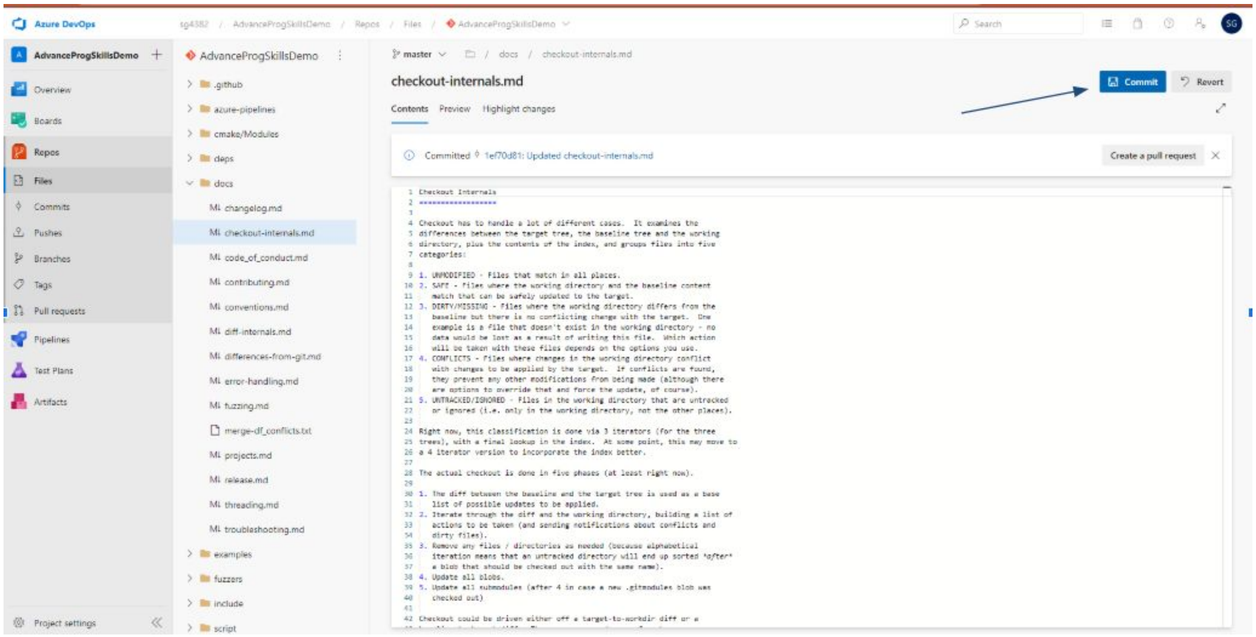
The current checkout implementation is based on a baseline-to-target diff.

4. First click on the Edit option on the top right corner. You can see the file changed to edit mode.



```
Checkout Internals and
==================

Checkout has to handle a lot of different cases.  It examines the
differences between the target tree, the baseline tree and the working
directory, plus the contents of the index, and groups files into five
categories:

1. UNMODIFIED - Files that match in all places.
2. SAFE - Files where the working directory and the baseline content
   match that can be safely updated to the target.
3. DIRTY/MISSING - Files where the working directory differs from the
   baseline but there is no conflicting change with the target.  One
   example is a file that doesn't exist in the working directory - no
   data would be lost as a result of writing this file.  Which action
   will be taken with these files depends on the options you use.
4. CONFLICTS - Files where changes in the working directory conflict
   with changes to be applied by the target.  If conflicts are found,
   they prevent any other modifications from being made (although there
   are options to override that and force the update, of course).
5. UNTRACKED/IGNORED - Files in the working directory that are untracked
   or ignored (i.e. only in the working directory, not the other places).

Right now, this classification is done via 3 iterators (for the three
trees), with a final lookup in the index.  At some point, this may move to
a 4 iterator version to incorporate the index better.

The actual checkout is done in five phases (at least right now).

1. The diff between the baseline and the target tree is used as a base
   list of possible updates to be applied.
2. Iterate through the diff and the working directory, building a list of
   actions to be taken (and sending notifications about conflicts and
   dirty files).
3. Remove any files / directories as needed (because alphabetical
   iteration means that an untracked directory will end up sorted *after*
   a blob that should be checked out with the same name).
4. Update all blobs.
5. Update all submodules (after 4 in case a new .gitmodules blob was
   checked out)

Checkout could be driven either off a target-to-workdir diff or a
```

5. Now I just clear the typo and click the Commit button on the top right corner.

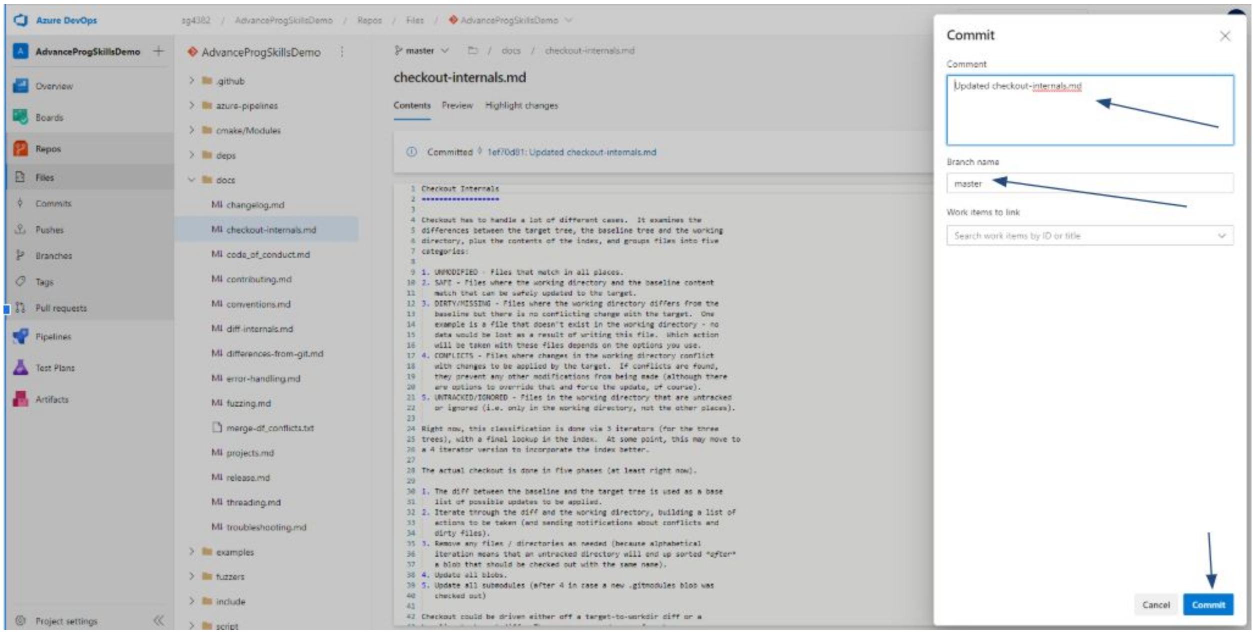


checkout-internals.md

Committed 1ef70d81: Updated checkout-internals.md

```
Checkout Internals
==================

Checkout has to handle a lot of different cases.  It examines the
differences between the target tree, the baseline tree and the working
directory, plus the contents of the index, and groups files into five
categories:

1. UNMODIFIED - Files that match in all places.
2. SAFE - Files where the working directory and the baseline content
   match that can be safely updated to the target.
3. DIRTY/MISSING - Files where the working directory differs from the
   baseline but there is no conflicting change with the target.  One
   example is a file that doesn't exist in the working directory - no
   data would be lost as a result of writing this file.  Which action
   will be taken with these files depends on the options you use.
4. CONFLICTS - Files where changes in the working directory conflict
   with changes to be applied by the target.  If conflicts are found,
   they prevent any other modifications from being made (although there
   are options to override that and force the update, of course).
5. UNTRACKED/IGNORED - Files in the working directory that are untracked
   or ignored (i.e. only in the working directory, not the other places).

Right now, this classification is done via 3 iterators (for the three
trees), with a final lookup in the index.  At some point, this may move to
a 4 iterator version to incorporate the index better.

The actual checkout is done in five phases (at least right now).

1. The diff between the baseline and the target tree is used as a base
   list of possible updates to be applied.
2. Iterate through the diff and the working directory, building a list of
   actions to be taken (and sending notifications about conflicts and
   dirty files).
3. Remove any files / directories as needed (because alphabetical
   iteration means that an untracked directory will end up sorted *after*
   a blob that should be checked out with the same name).
4. Update all blobs.
5. Update all submodules (after 4 in case a new .gitmodules blob was
   checked out)

Checkout could be driven either off a target-to-workdir diff or a
```

6. It asks me which branch to commit and a comment option.



checkout-internals.md

Committed 1ef70d81: Updated checkout-internals.md

Commit

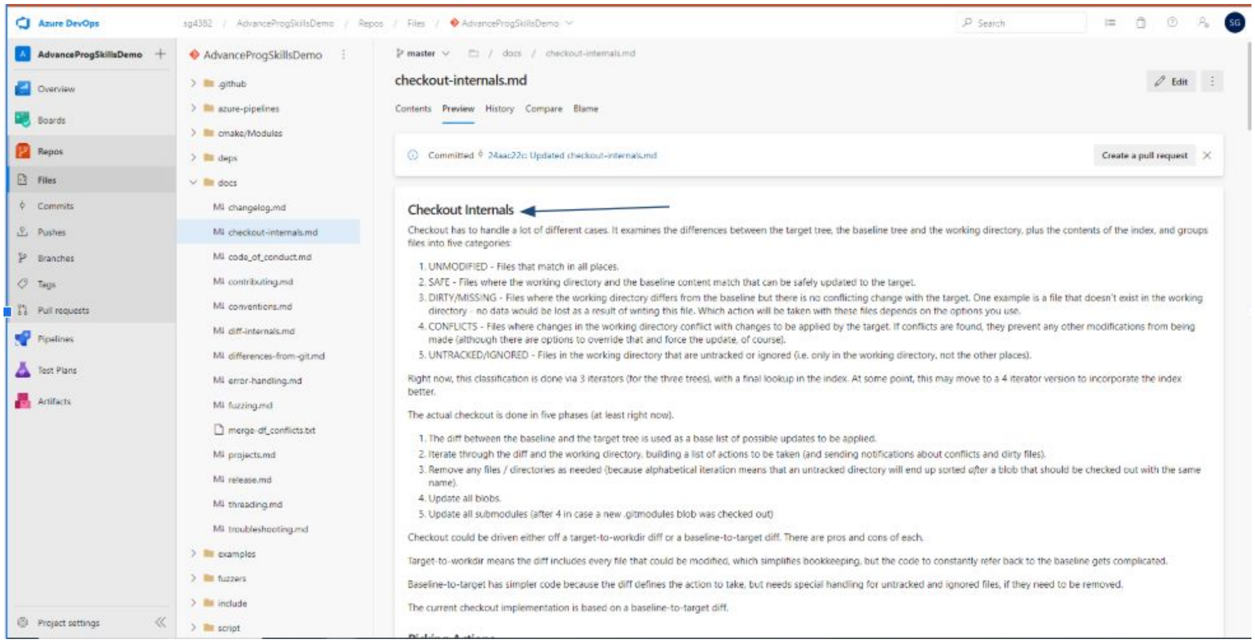
Comment: Updated checkout-internals.md

Branch name: master

Work items to link: Search work items by ID or title

```
Checkout Internals
==================

Checkout has to handle a lot of different cases.  It examines the
differences between the target tree, the baseline tree and the working
directory, plus the contents of the index, and groups files into five
categories:

1. UNMODIFIED - Files that match in all places.
2. SAFE - Files where the working directory and the baseline content
   match that can be safely updated to the target.
3. DIRTY/MISSING - Files where the working directory differs from the
   baseline but there is no conflicting change with the target.  One
   example is a file that doesn't exist in the working directory - no
   data would be lost as a result of writing this file.  Which action
   will be taken with these files depends on the options you use.
4. CONFLICTS - Files where changes in the working directory conflict
   with changes to be applied by the target.  If conflicts are found,
   they prevent any other modifications from being made (although there
   are options to override that and force the update, of course).
5. UNTRACKED/IGNORED - Files in the working directory that are untracked
   or ignored (i.e. only in the working directory, not the other places).

Right now, this classification is done via 3 iterators (for the three
trees), with a final lookup in the index.  At some point, this may move to
a 4 iterator version to incorporate the index better.

The actual checkout is done in five phases (at least right now).

1. The diff between the baseline and the target tree is used as a base
   list of possible updates to be applied.
2. Iterate through the diff and the working directory, building a list of
   actions to be taken (and sending notifications about conflicts and
   dirty files).
3. Remove any files / directories as needed (because alphabetical
   iteration means that an untracked directory will end up sorted *after*
   a blob that should be checked out with the same name).
4. Update all blobs.
5. Update all submodules (after 4 in case a new .gitmodules blob was
   checked out)

Checkout could be driven either off a target-to-workdir diff or a
```

7. Now you can see the updated file. This is how we commit to the master branch. Committing to a master branch is very simple, but it is not sensible to do so. The code or any document should go through at least 1 code review. Here where repos come to save the project. You have to open a pull request so that other developers can see the changes you made.



checkout-internals.md

Committed 24aac22c: Updated checkout-internals.md

Checkout Internals

Checkout has to handle a lot of different cases. It examines the differences between the target tree, the baseline tree and the working directory, plus the contents of the index, and groups files into five categories:

1. UNMODIFIED - Files that match in all places.
2. SAFE - Files where the working directory and the baseline content match that can be safely updated to the target.
3. DIRTY/MISSING - Files where the working directory differs from the baseline but there is no conflicting change with the target. One example is a file that doesn't exist in the working directory - no data would be lost as a result of writing this file. Which action will be taken with these files depends on the options you use.
4. CONFLICTS - Files where changes in the working directory conflict with changes to be applied by the target. If conflicts are found, they prevent any other modifications from being made (although there are options to override that and force the update, of course).
5. UNTRACKED/IGNORED - Files in the working directory that are untracked or ignored (i.e. only in the working directory, not the other places).

Right now, this classification is done via 3 iterators (for the three trees), with a final lookup in the index. At some point, this may move to a 4 iterator version to incorporate the index better.

The actual checkout is done in five phases (at least right now).

1. The diff between the baseline and the target tree is used as a base list of possible updates to be applied.
2. Iterate through the diff and the working directory, building a list of actions to be taken (and sending notifications about conflicts and dirty files).
3. Remove any files / directories as needed (because alphabetical iteration means that an untracked directory will end up sorted *after* a blob that should be checked out with the same name).
4. Update all blobs.
5. Update all submodules (after 4 in case a new .gitmodules blob was checked out)

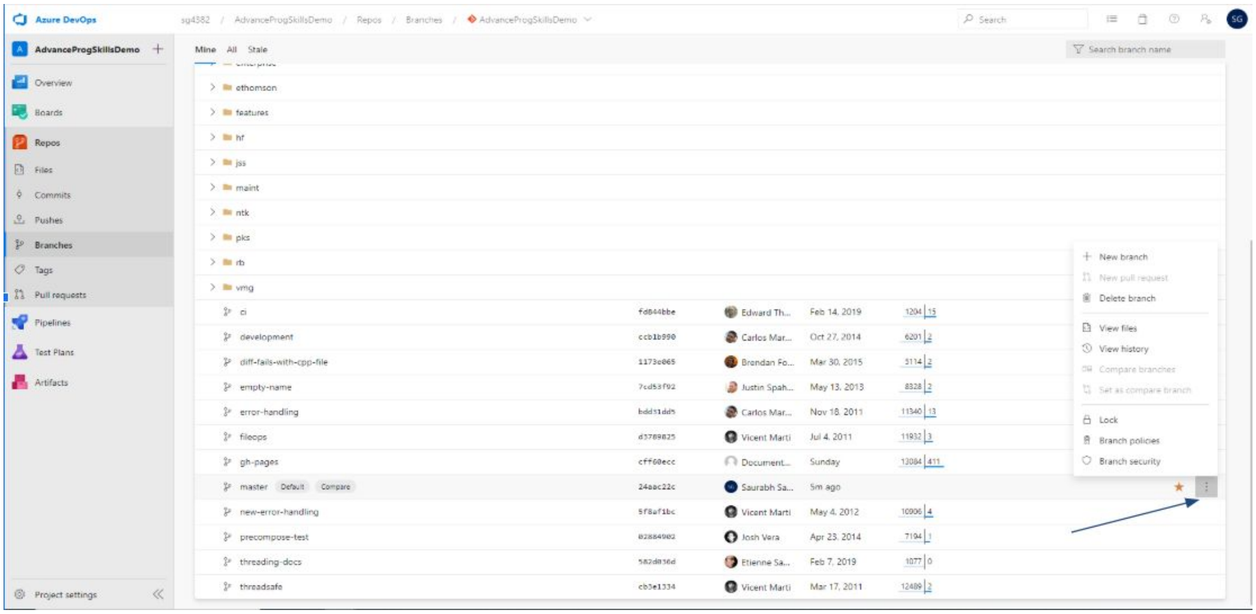
Checkout could be driven either off a target-to-workdir diff or a baseline-to-target diff. There are pros and cons of each.

Target-to-workdir means the diff includes every file that could be modified, which simplifies bookkeeping, but the code to constantly refer back to the baseline gets complicated.

Baseline-to-target has simpler code because the diff defines the action to take, but needs special handling for untracked and ignored files, if they need to be removed.

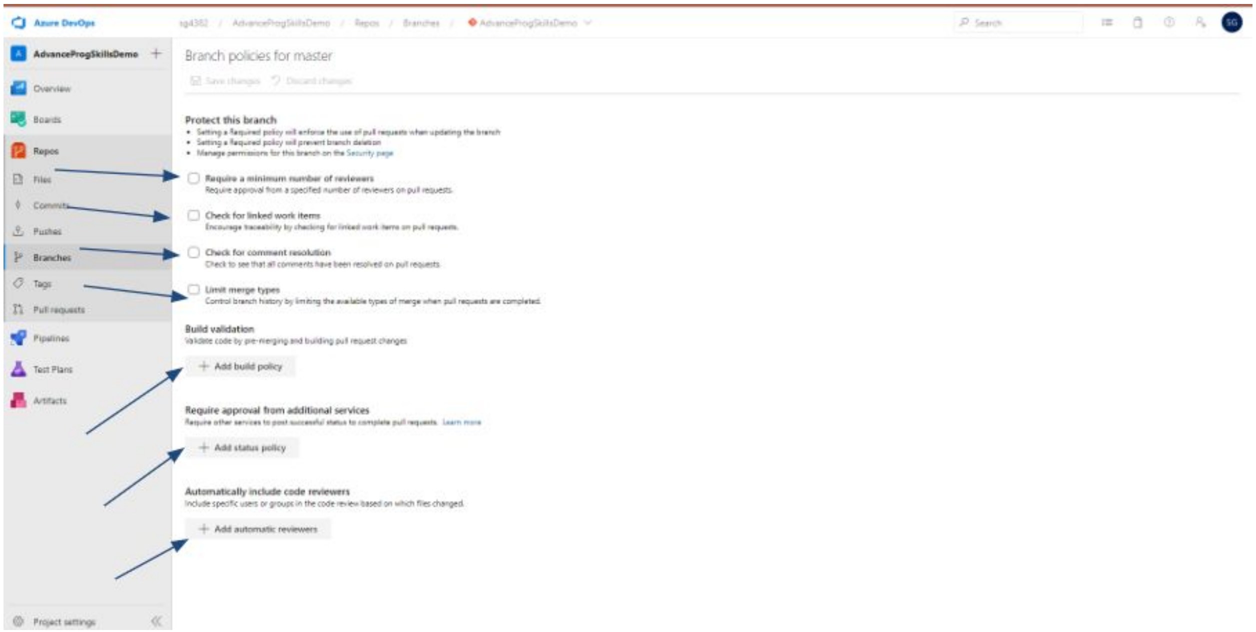
The current checkout implementation is based on a baseline-to-target diff.

8. Let us see how it is done. Click on Branches and drag the cursor to three dots at the end of the master branch option. You can see these options.



ethomson				
features				
hf				
jss				
maint				
ntk				
pks				
rb				
vmg				
ci	fd844bbe	Edward Th...	Feb 14, 2019	1204 | 15
development	ccb1b990	Carlos Mar...	Oct 27, 2014	6201 | 2
diff-fails-with-cpp-file	1173e065	Brendan Fo...	Mar 30, 2015	5114 | 2
empty-name	7cd53f92	Justin Spah...	May 13, 2013	8328 | 2
error-handling	bdd51dd5	Carlos Mar...	Nov 18, 2011	11340 | 13
fileops	d3789825	Vicent Marti	Jul 4, 2011	11932 | 3
gh-pages	cff60ecc	Document...	Sunday	13084 | 411
master Default Compare	24aac22c	Saurabh Sa...	5m ago	
new-error-handling	5f8af1bc	Vicent Marti	May 4, 2012	10906 | 4
precompose-test	82884982	Josh Vera	Apr 23, 2014	7194 | 1
threading-docs	582d836d	Etienne Sa...	Feb 7, 2019	1077 | 0
threadsafe	cb3e1334	Vicent Marti	Mar 17, 2011	12489 | 2

9. Click on branch policies.



Branch policies for master

Protect this branch

- Setting a Required policy will enforce the use of pull requests when updating the branch
- Setting a Required policy will prevent branch deletion
- Manage permissions for this branch on the Security page

☐ Require a minimum number of reviewers
Require approval from a specified number of reviewers on pull requests.

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

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

☐ Limit merge types
Control branch history by limiting the available types of merge when pull requests are completed.

Build validation
Validate code by pre-merging and building pull request changes

+ Add build policy

Require approval from additional services
Require other services to post successful status to complete pull requests. Learn more

+ Add status policy

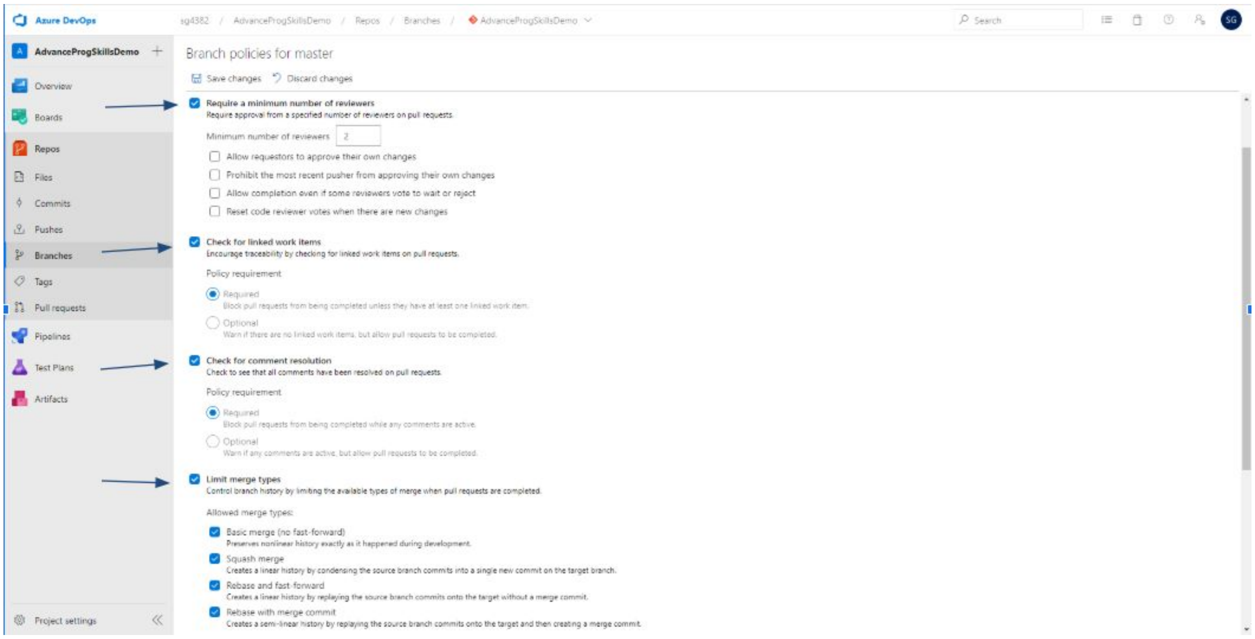
Automatically include code reviewers
Include specific users or groups in the code review based on which files changed.

+ Add automatic reviewers

There are a lot of options like such as,

- **Require a minimum number of reviewers:** Requires a minimum number of approvals from reviewers.
- **Check for linked work items:** Traces and has a linked story/task on board. This keeps full traceability.
- **Check for comment resolution:** To see all comments have been resolved on pull requests.
- **Limit merge types:** Control branching history by limiting the types of merges.
- **Add Build policy:** Validate code by building with pull request changes, after every pull request it should go through a pipeline.
- **Add status policy:** Requires other services post successful pull requests.
- **Add automatic reviewers:** Include specific users to review the changes in the code.

10. These are the options we can see when we enable each checkbox.



Branch policies for master

☑ **Require a minimum number of reviewers**
Require approval from a specified number of reviewers on pull requests.

Minimum number of reviewers: 2

☐ 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
☐ Reset code reviewer votes when there are new changes

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

Policy requirement

● 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.

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

Policy requirement

● 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.

☑ **Limit merge types**
Control branch history by limiting the available types of merge when pull requests are completed.

Allowed merge types:

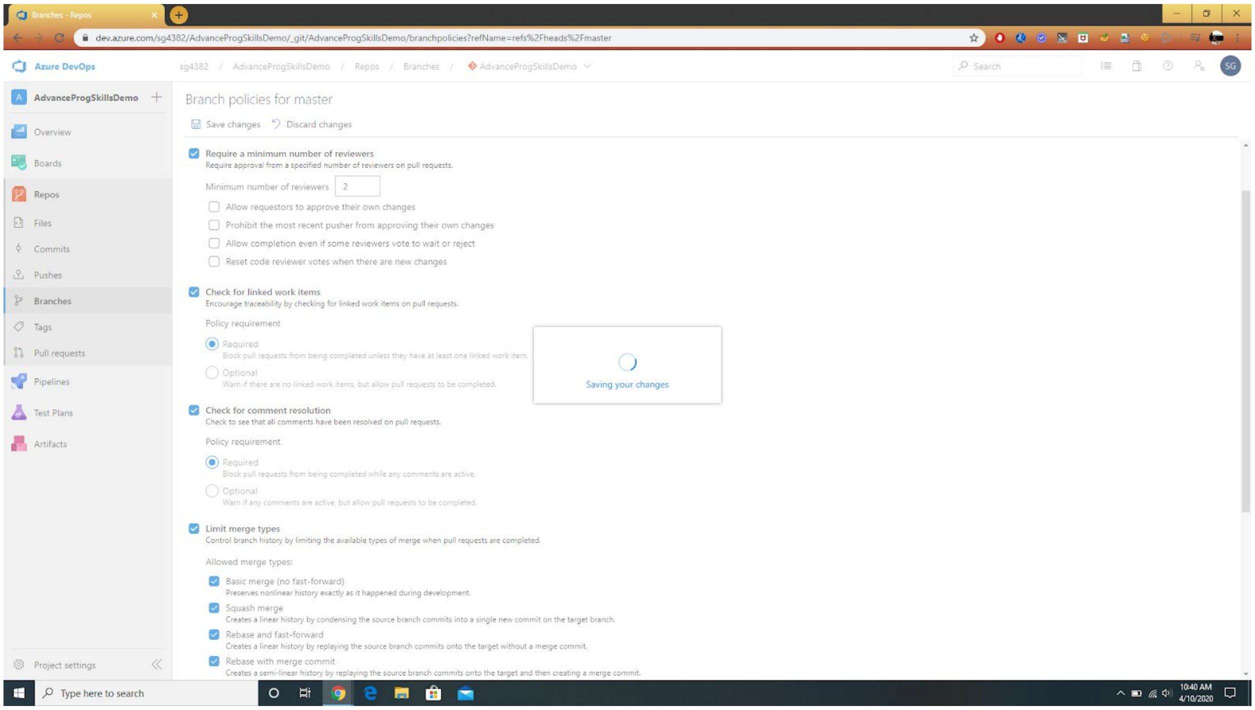
☑ Basic merge (no fast-forward)
Preserves nonlinear history exactly as it happened during development.

☑ Squash merge
Creates a linear history by condensing the source branch commits into a single new commit on the target branch.

☑ Rebase and fast-forward
Creates a linear history by replaying the source branch commits onto the target without a merge commit.

☑ Rebase with merge commit
Creates a semi-linear history by replaying the source branch commits onto the target and then creating a merge commit.

11. Select the options you want to have in your project and click on Save changes.



dev.azure.com/sg4382/AdvanceProgSkillsDemo/_git/AdvanceProgSkillsDemo/branchpolicies?refName=refs%2Fheads%2Fmaster

Saving your changes

Branch policies for master

☑ Require a minimum number of reviewers
Require approval from a specified number of reviewers on pull requests.

Minimum number of reviewers: 2

☐ 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
☐ Reset code reviewer votes when there are new changes

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

Policy requirement

● 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.

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

Policy requirement

● 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.

☑ Limit merge types
Control branch history by limiting the available types of merge when pull requests are completed.

Allowed merge types:

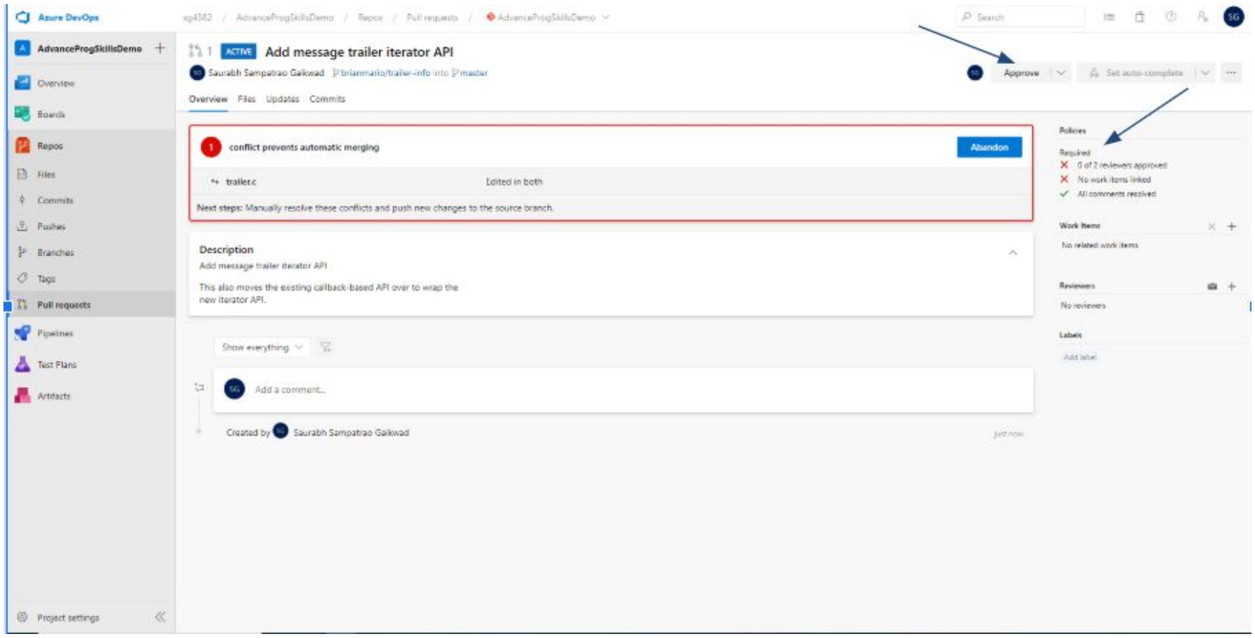
☑ Basic merge (no fast-forward)
Preserves nonlinear history exactly as it happened during development.

☑ Squash merge
Creates a linear history by condensing the source branch commits into a single new commit on the target branch.

☑ Rebase and fast-forward
Creates a linear history by replaying the source branch commits onto the target without a merge commit.

☑ Rebase with merge commit
Creates a semi-linear history by replaying the source branch commits onto the target and then creating a merge commit.

12. Later I created a pull request to see how we can approve the pull request and all the criteria are seen. Here, we can see individual Files, Updates and each Commit done by the author.



1 ACTIVE Add message trailer iterator API

Saurabh Sampatrao Gaikwad brianmario/trailer-info into master

conflict prevents automatic merging

trailer.c Edited in both

Next steps: Manually resolve these conflicts and push new changes to the source branch.

Description

Add message trailer iterator API

This also moves the existing callback-based API over to wrap the new iterator API.

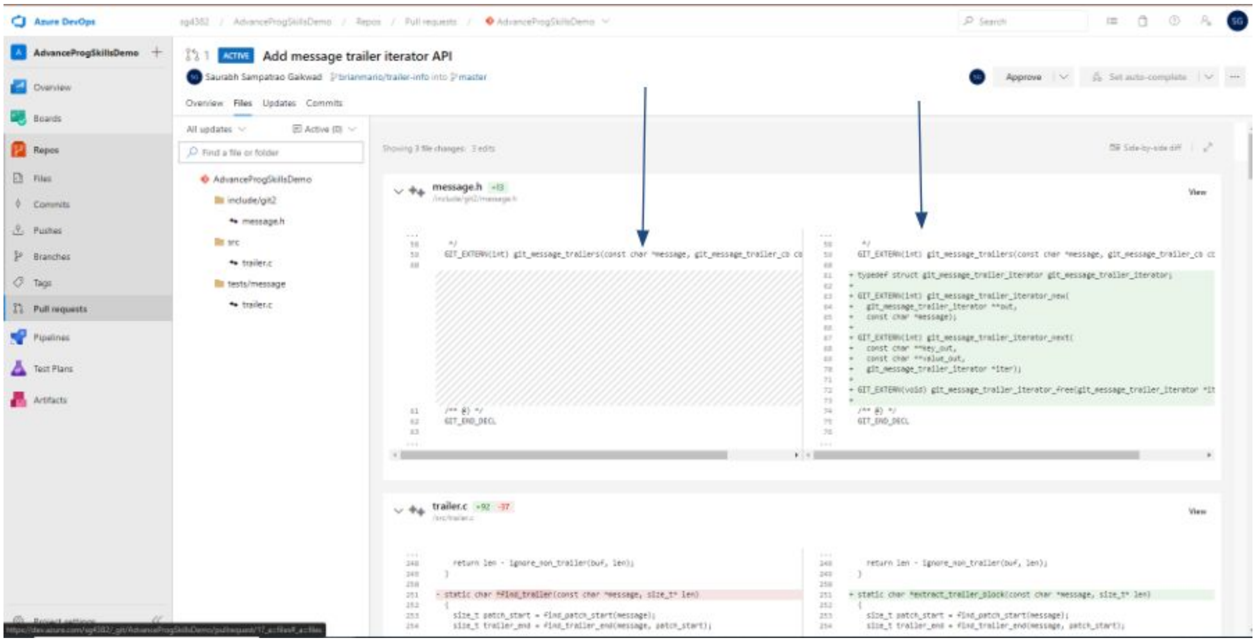
Created by Saurabh Sampatrao Gaikwad

Required
- ✗ 0 of 2 reviewers approved
- ✗ No work items linked
- ✓ All comments resolved

No related work items

No reviewers

13. Files view



1 ACTIVE Add message trailer iterator API

Saurabh Sampatrao Gaikwad brianmario/trailer-info into master

AdvanceProgSkillsDemo
- include/git2
  - message.h
- src
  - trailer.c
- tests/message
  - trailer.c

Showing 3 file changes: 3 edits

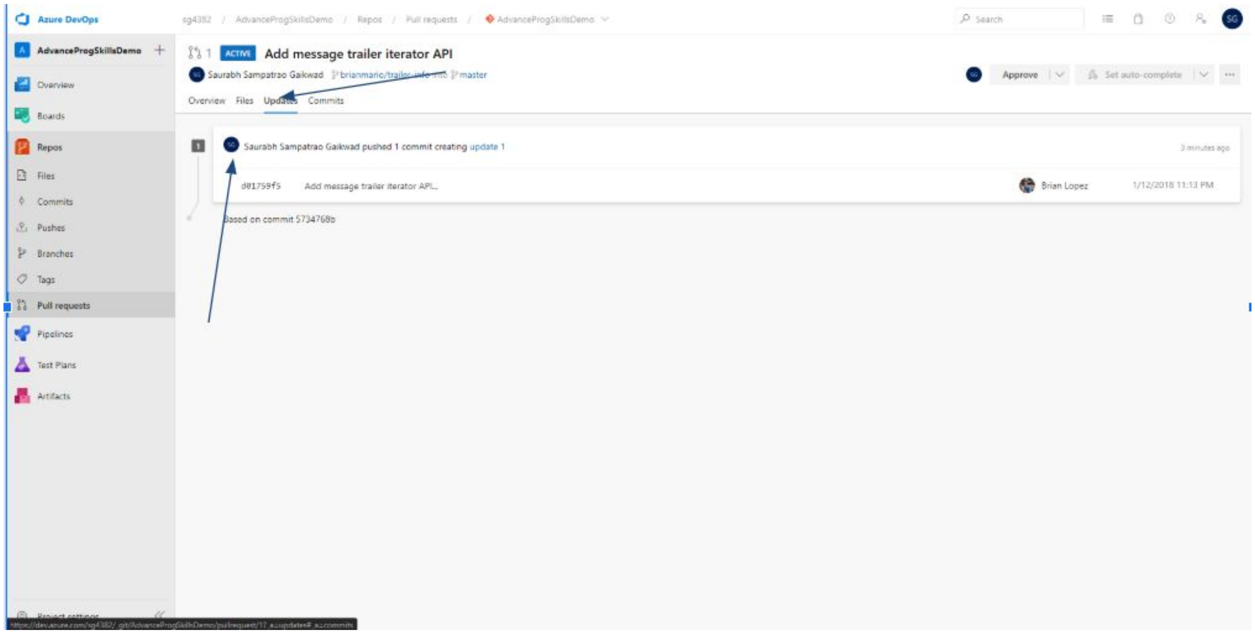
message.h +13 /include/git2/message.h

```
 */
GIT_EXTERN(int) git_message_trailers(const char *message, git_message_trailer_cb cc

+ typedef struct git_message_trailer_iterator git_message_trailer_iterator;
+
+ GIT_EXTERN(int) git_message_trailer_iterator_new(
+   git_message_trailer_iterator **out,
+   const char *message);
+
+ GIT_EXTERN(int) git_message_trailer_iterator_next(
+   const char **key_out,
+   const char **value_out,
+   git_message_trailer_iterator *iter);
+
+ GIT_EXTERN(void) git_message_trailer_iterator_free(git_message_trailer_iterator *it
+
/** @} */
GIT_END_DECL
```

trailer.c +92 -37 /src/trailer.c

```
  return len - ignore_non_trailer(buf, len);
}

- static char *find_trailer(const char *message, size_t* len)
+ static char *extract_trailer_block(const char *message, size_t* len)
{
  size_t patch_start = find_patch_start(message);
  size_t trailer_end = find_trailer_end(message, patch_start);
```

14. Updates



1 ACTIVE Add message trailer iterator API

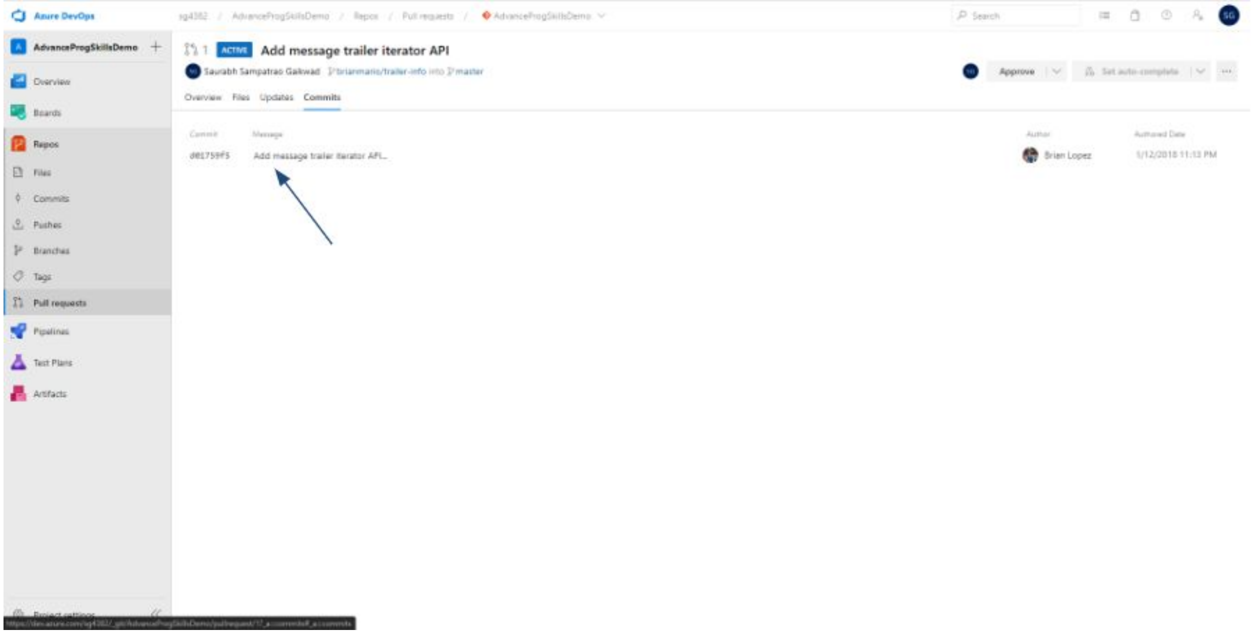
Saurabh Sampatrao Gaikwad brianmario/trailer-info into master

Saurabh Sampatrao Gaikwad pushed 1 commit creating update 1 — 3 minutes ago

d01759f5 Add message trailer iterator API... Brian Lopez 1/12/2018 11:13 PM

Based on commit 5734768b

15. Commits



1 ACTIVE Add message trailer iterator API

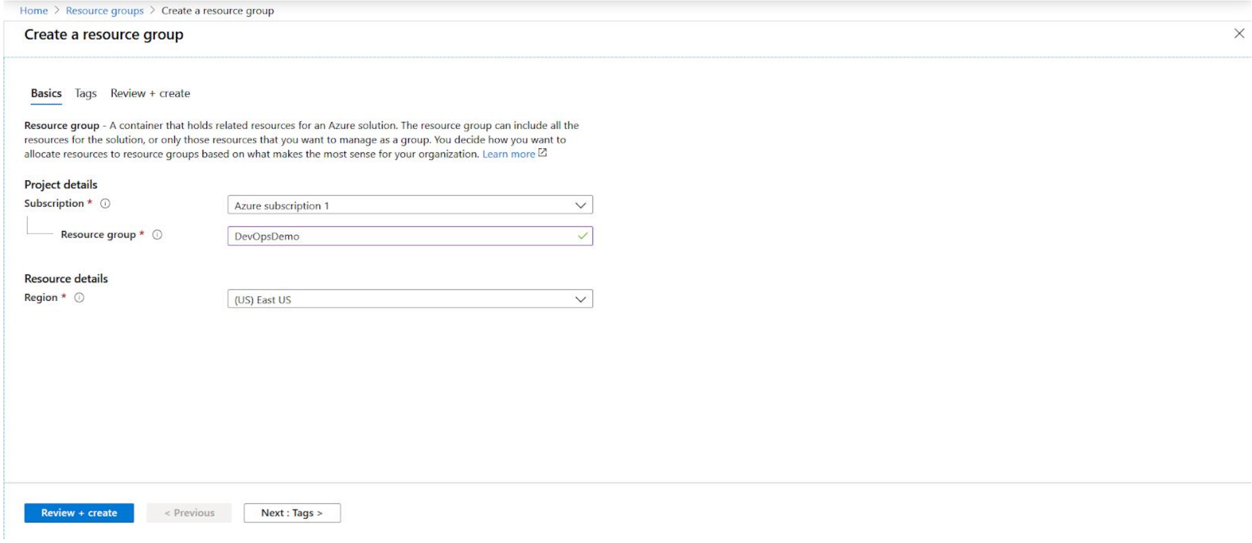
Saurabh Sampatrao Gaikwad brianmario/trailer-info into master

Commit	Message	Author	Authored Date
d01759f5	Add message trailer iterator API...	Brian Lopez	1/12/2018 11:13 PM

When all the policies are covered, (which can be seen on the right top corner in point 12 of repos) the pull request is approved.

Presentation Demo:
Creating a CI/CD Pipeline for Github Project and release it on Azure Cloud

1. Sign in to the Azure portal from this link.
2. In the search tab, search for Resource Group and click on +Add. Create a resource group for the project. It is a good practice to create a new resource group for every project. It helps in organizing and easily maintaining project permissions.



Home > Resource groups > Create a resource group

Create a resource group

Basics | Tags | Review + create

Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Learn more

Project details

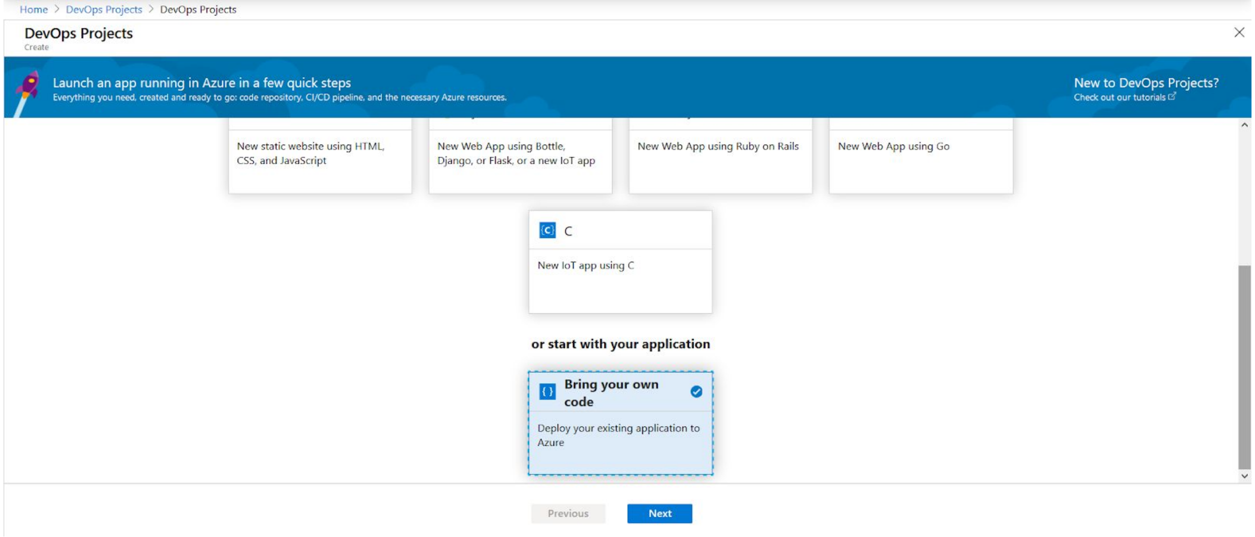
Subscription * : Azure subscription 1

Resource group * : DevOpsDemo

Resource details

Region * : (US) East US

3. Search for DevOps Project service from the search bar and click on +Add to create a new project. We can either create a new project based on the different environments listed like Node.js, .NET, etc. or use the existing code. We will use the Github repository as a codebase for this Demo. Select “Bring your own code” and follow the steps.



Home > DevOps Projects > DevOps Projects

DevOps Projects
Create

Launch an app running in Azure in a few quick steps
Everything you need, created and ready to go: code repository, CI/CD pipeline, and the necessary Azure resources.

New to DevOps Projects?
Check out our tutorials

New static website using HTML, CSS, and JavaScript

New Web App using Bottle, Django, or Flask, or a new IoT app

New Web App using Ruby on Rails

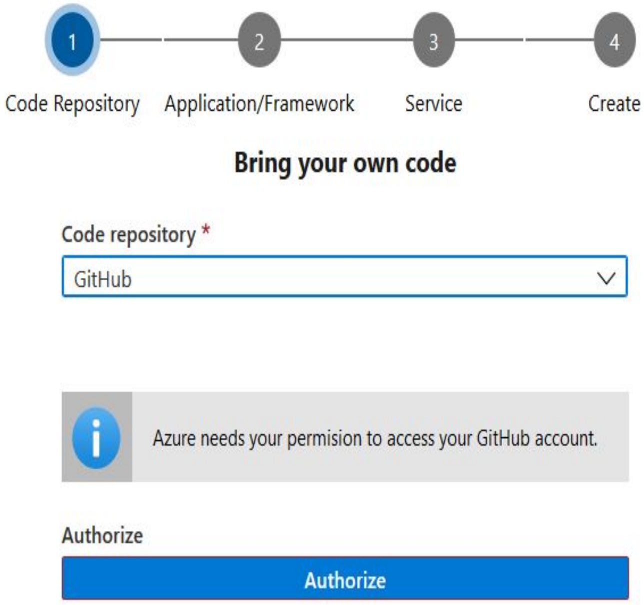
New Web App using Go

C
New IoT app using C

or start with your application

Bring your own code
Deploy your existing application to Azure

4. Select Github as a code repository. Fill out the basic project details and select the repository to use from Github. Follow through instructions click Done.



Bring your own code

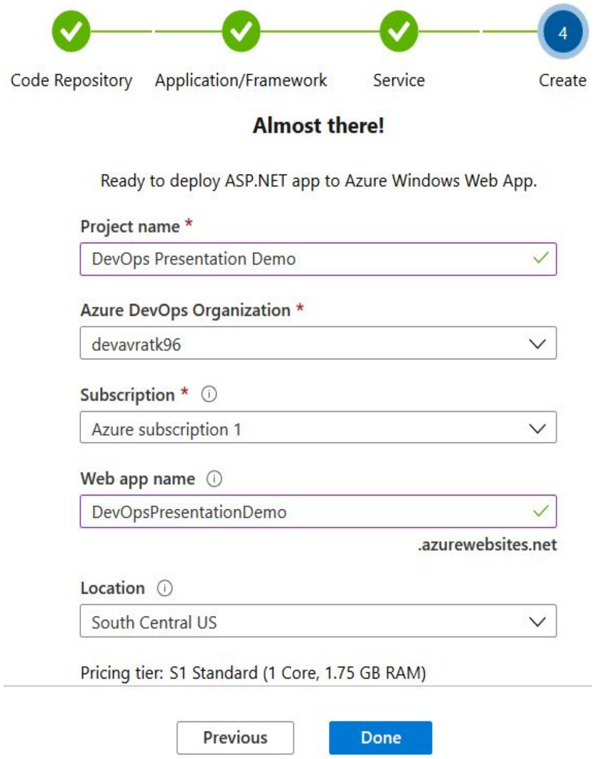
Code repository *

GitHub

Azure needs your permision to access your GitHub account.

Authorize

Authorize



Almost there!

Ready to deploy ASP.NET app to Azure Windows Web App.

Project name *: DevOps Presentation Demo

Azure DevOps Organization *: devavratk96

Subscription *: Azure subscription 1

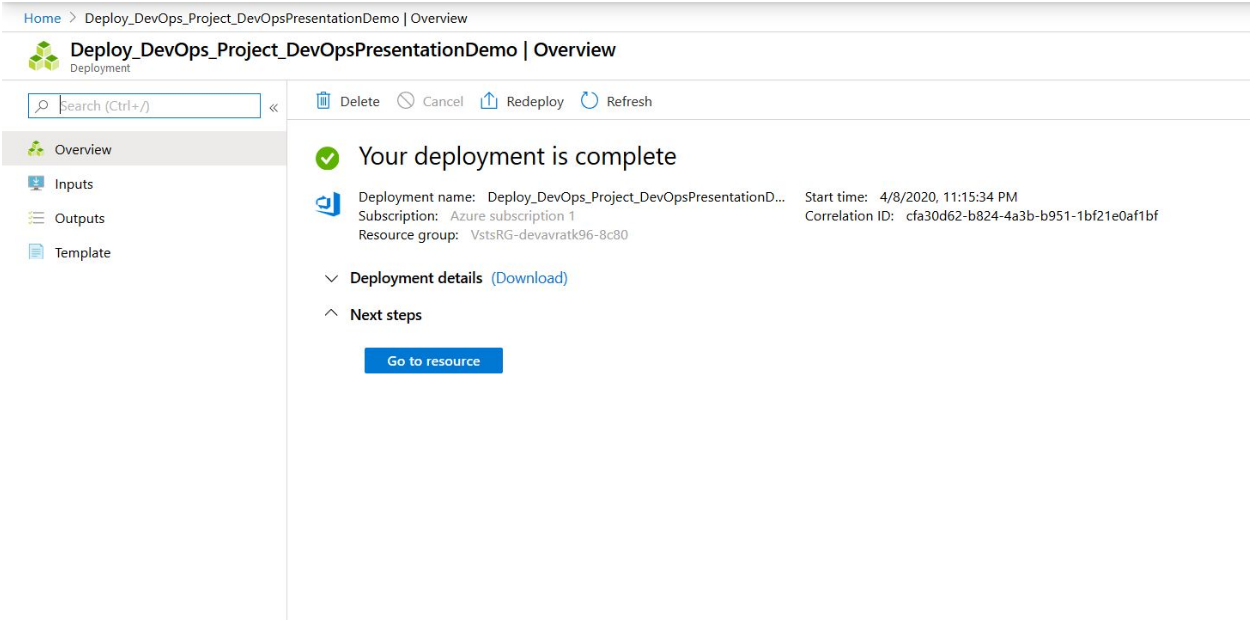
Web app name: DevOpsPresentationDemo .azurewebsites.net

Location: South Central US

Pricing tier: S1 Standard (1 Core, 1.75 GB RAM)

Previous | Done

5. Once everything is set, Azure will take time to create an environment for the project and start building the project. Click on “Go to resources”.



Deploy_DevOps_Project_DevOpsPresentationDemo | Overview

Deployment

Your deployment is complete

Deployment name: Deploy_DevOps_Project_DevOpsPresentationD...
Subscription: Azure subscription 1
Resource group: VstsRG-devavratk96-8c80

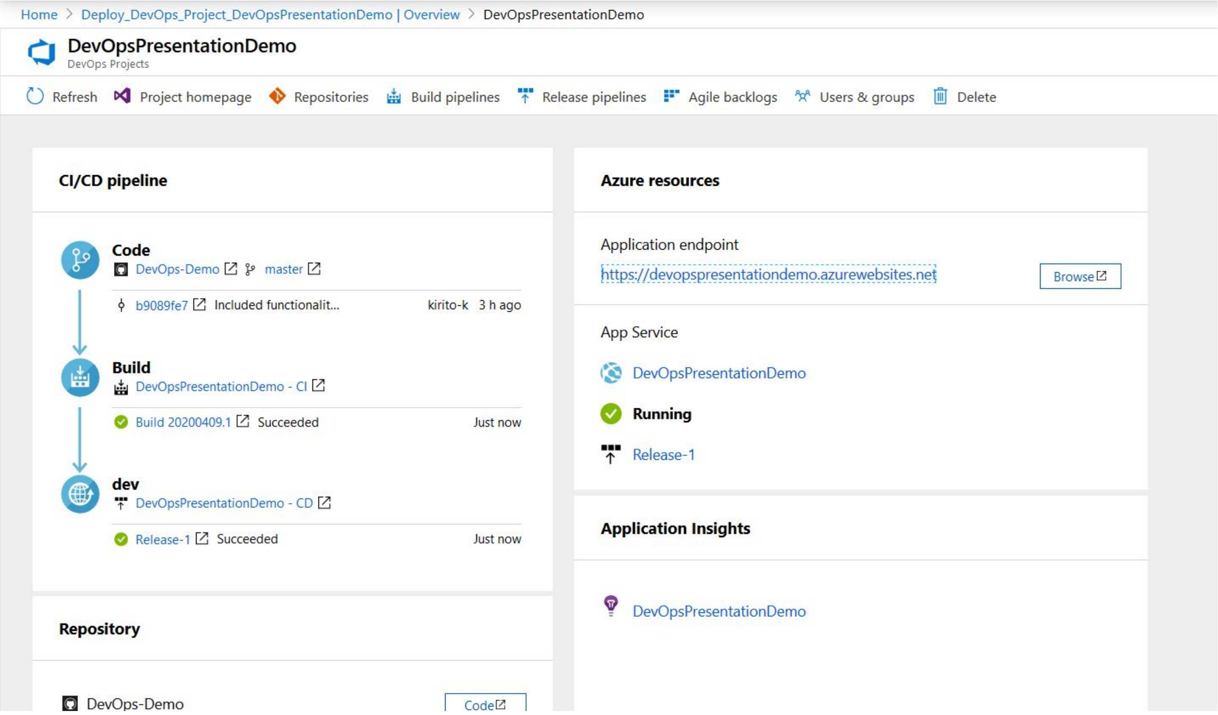
Start time: 4/8/2020, 11:15:34 PM
Correlation ID: cfa30d62-b824-4a3b-b951-1bf21e0af1bf

Deployment details (Download)

Next steps

Go to resource

6. We can see the progress of every stage i.e. Code, Build, Dev while it is in progress. The Code stage pulls the project from the Github to Azure. The Build phase starts the build process for the code. It generates dependencies and a relevant environment where code can be in a release-ready state. In the Dev phase release the code on Azure cloud so that it can be accessed through URL. URL is mentioned as “Application endpoint”.



DevOpsPresentationDemo
DevOps Projects

CI/CD pipeline

Code
DevOps-Demo master
b9089fe7 Included functionalit... kirito-k 3 h ago

Build
DevOpsPresentationDemo - CI
Build 20200409.1 Succeeded Just now

dev
DevOpsPresentationDemo - CD
Release-1 Succeeded Just now

Repository

DevOps-Demo

Azure resources

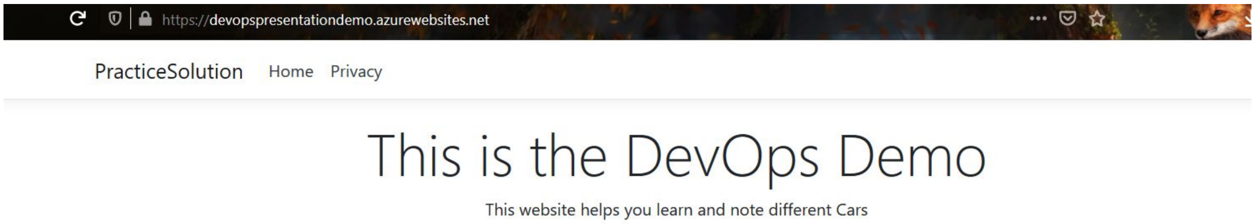
Application endpoint
https://devopspresentationdemo.azurewebsites.net

App Service
DevOpsPresentationDemo
Running
Release-1

Application Insights

DevOpsPresentationDemo

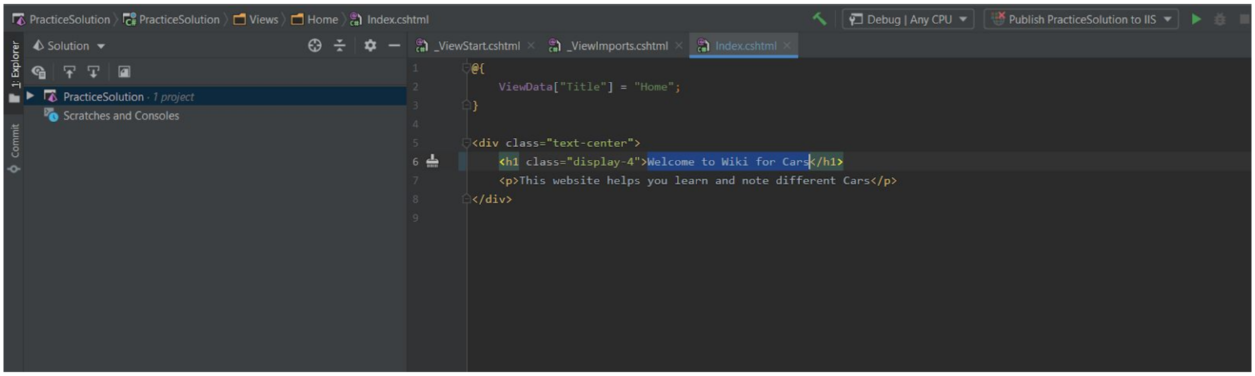
7. Once all stages are complete we can access this website using the Application endpoint link.



This is the DevOps Demo

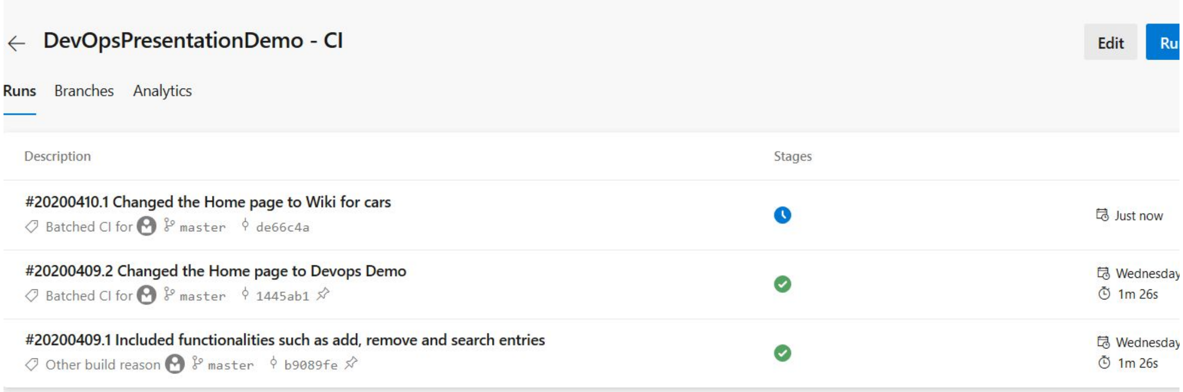
This website helps you learn and note different Cars

8. To check whether the pipeline is working properly Change the codebase locally and push it to Github.

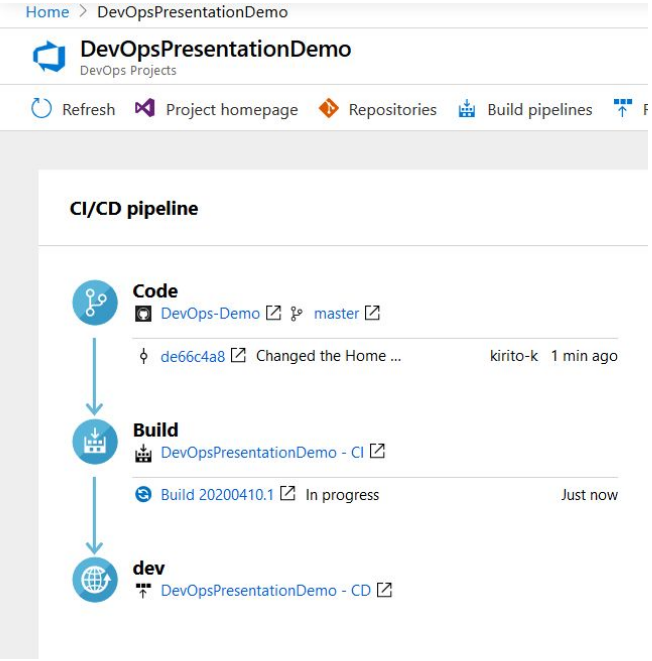

PracticeSolution 〉 PracticeSolution 〉 Views 〉 Home 〉 Index.cshtml
Debug | Any CPU
Publish PracticeSolution to IIS
Solution
PracticeSolution · 1 project
Scratches and Consoles
_ViewStart.cshtml
_ViewImports.cshtml
Index.cshtml
1: Explorer
Commit

```
@{
    ViewData["Title"] = "Home";
}

<div class="text-center">
    <h1 class="display-4">Welcome to Wiki for Cars</h1>
    <p>This website helps you learn and note different Cars</p>
</div>
```

9. Once the changes are pushed on Github, we can see the changes are in progress on our Azure DevOps Project Portal and Pipelines in organizations. This will run all the phases which we discussed before to see everything is working correctly.



Description	Stages	
#20200410.1 Changed the Home page to Wiki for cars Batched CI for master de66c4a		Just now
#20200409.2 Changed the Home page to Devops Demo Batched CI for master 1445ab1	✓	Wednesday 1m 26s
#20200409.1 Included functionalities such as add, remove and search entries Other build reason master b9089fe	✓	Wednesday 1m 26s



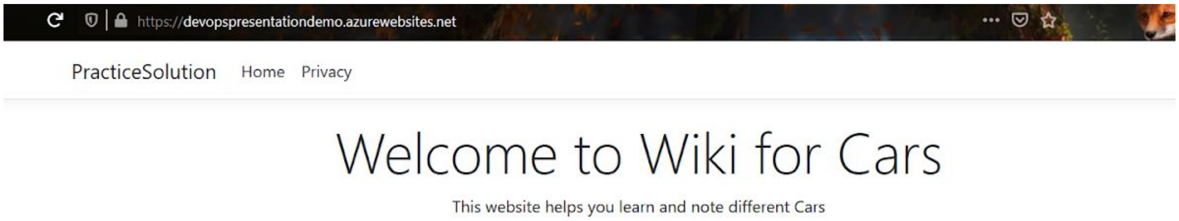
CI/CD pipeline

Code
DevOps-Demo master
de66c4a8 Changed the Home ... kirito-k 1 min ago

Build
DevOpsPresentationDemo - CI
Build 20200410.1 In progress Just now

dev
DevOpsPresentationDemo - CD

10. After everything is set, the results will be reflected directly on our web URL.



https://devopspresentationdemo.azurewebsites.net

Welcome to Wiki for Cars

This website helps you learn and note different Cars

Conclusion

In a nutshell, after a project is created by a team of developers and released, they need to assign someone for managing it. The operations team has to handle this task and maintain the codebase. That's where DevOps shins. For companies, it is essential that when any changes are made, those are well tested and rapidly integrated into the new releases. This process is automated using the Azure CI/CD Pipeline phase of DevOps. Different teams require isolated environments that are handled by Azure Repos. These development environments may need their own package configurations which can be handled by Azure Artifacts as a package manager. Last but not least the development cycle itself can be managed using Azure Boards for better organization.

Overall Azure DevOps provides all the tools necessary for the entire software development life cycle written by any size of the team in any language on any platform.