

DevSecOps, day 2



2. Lab: Tasks completed

You've done some work

- Yesterday, you already completed a few stories!
 - You installed your Dev Workstations.
 - You cloned the Git repositories.
 - You got the local JuiceShop to run.

Assignment

- Make a task, per team member, for these stories:
 - Setup Lab VM
 - Clone Git repo
 - Run local test.
- Set the tasks AND the stories to "Done".

For example

- From:

The screenshot shows a Jira team board for 'ITVitae team 1 Team'. The board is configured with a Kanban workflow and is currently in 'Sprint 1'. The columns are 'To Do', 'Doing', and 'Done'. The board displays several work items with their IDs, titles, assignees, and states.

Item ID	Title	Assignee	State
131	Setup lab VM	Unassigned	To Do
132	Clone Git repo	Unassigned	To Do
137	Ruben	student1	To Do
138	Tabitha	student2	To Do
140	Ruben	student1	To Do
141	Tabitha	student2	To Do
136	Tess	Tess Sluijter	Doing
139	Tess	Tess Sluijter	Doing

For example

- To:

The screenshot shows a Jira team board for 'ITVitae team 1 Team'. The board is set to 'Sprint 1' and 'Person: All'. It displays a Kanban workflow with three columns: 'To Do', 'Doing', and 'Done'. On the left, there is a 'Collapse all' button and a list of items. Two items are currently in the 'To Do' column: '131 Setup lab VM' and '132 Clone Git repo', both assigned to 'Unassigned' and marked as 'Done'. The 'Doing' column is empty. The 'Done' column contains three items: '136 Tess' (assigned to 'Tess Sluijter'), '137 Ruben' (assigned to 'student1'), and '138 Tabitha' (assigned to 'student2'). All items in the 'Done' column are marked as 'Done'. The board also shows a '+ New Work Item' button and various filters and settings icons.

3. Cooperative software delivery



Preventing case mixups

All Repositories

Repositories Settings **Policies** Security

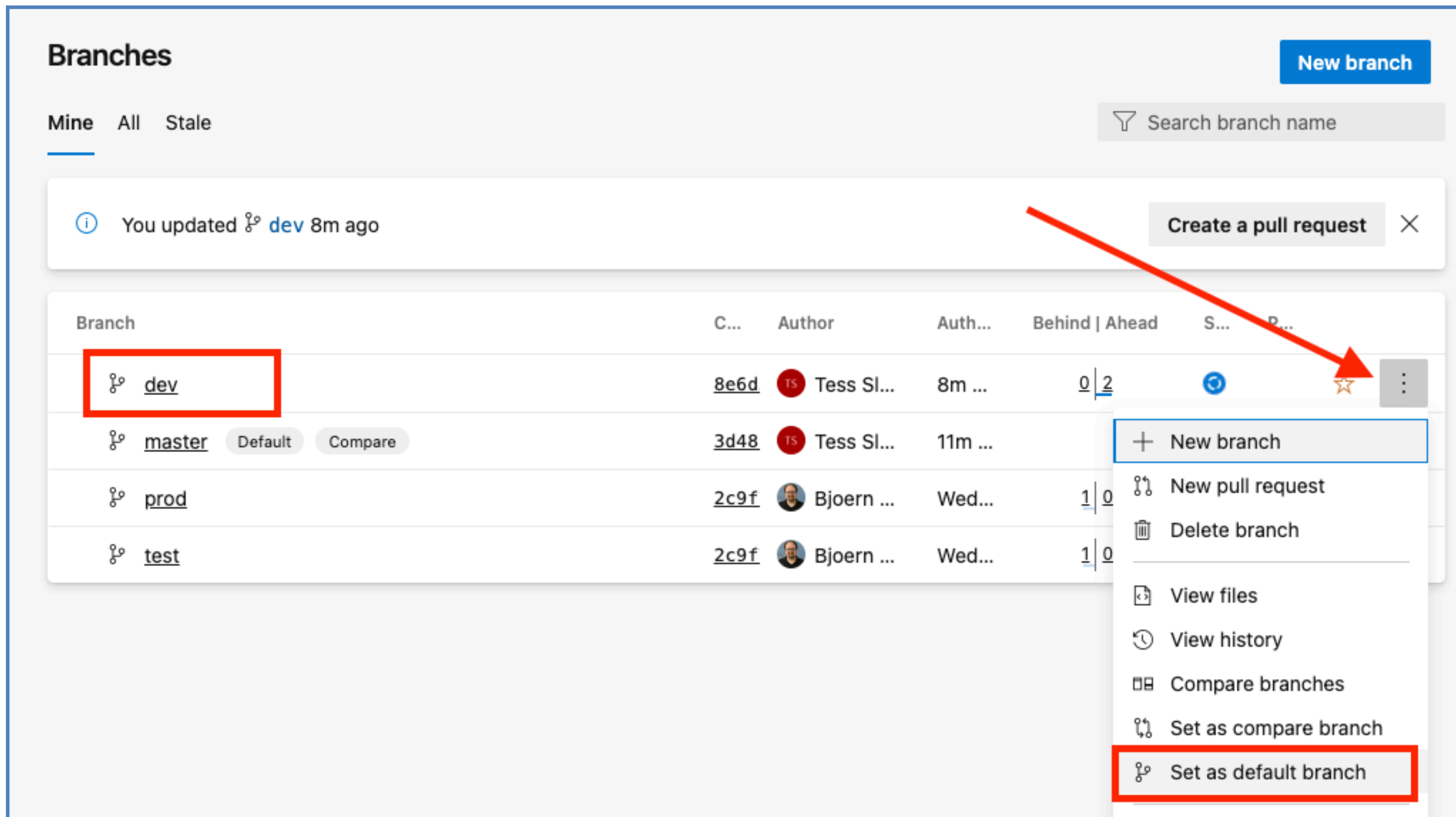
Repository Policies

<input type="checkbox"/> Off	Commit author email validation Block pushes with a commit author email that does not match the following patterns.
<input type="checkbox"/> Off	File path validation Block pushes from introducing file paths that match the following patterns.
<input checked="" type="checkbox"/> On	Case enforcement Avoid case-sensitivity conflicts by blocking pushes that change name casing on files, folders, branches, and tags. Learn more

Branching

- Go to your team's Git repository.
 - Go to "branches" in the side menu.
 - Make three new branches, from "main".
 - Make: *dev*, *test*, *prod*.

Set "dev" as default



The screenshot shows the GitHub 'Branches' page for a repository. The 'dev' branch is highlighted with a red box. A red arrow points from the 'dev' branch row to the 'Set as default branch' option in the dropdown menu. The dropdown menu is open, showing options like 'New branch', 'New pull request', 'Delete branch', 'View files', 'View history', 'Compare branches', 'Set as compare branch', and 'Set as default branch'. The 'Set as default branch' option is also highlighted with a red box.

Branches New branch

Mine All Stale Search branch name

You updated `dev` 8m ago Create a pull request

Branch	C...	Author	Auth...	Behind Ahead	S...	P...
<code>dev</code>	<code>8e6d</code>	Tess Sl...	8m ...	0 2		
<code>master</code> Default Compare	<code>3d48</code>	Tess Sl...	11m ...			
<code>prod</code>	<code>2c9f</code>	Bjoern ...	Wed...	1 0		
<code>test</code>	<code>2c9f</code>	Bjoern ...	Wed...	1 0		

- + New branch
- New pull request
- Delete branch
- View files
- View history
- Compare branches
- Set as compare branch
- Set as default branch**

Branching

- A quick demo:

```
$ cd ~/Documents/Team1JS  
$ git fetch  
$ git branch -r  
$ git switch main  
$ git switch dev
```

Let's cause problems

- Every team member does this at the same time.

```
$ git branch  
$ git switch dev  
$ nano MyFile.txt  
$ git add MyFile.txt  
$ git commit -m "Made my file."  
$ git push
```

Let's cause problems

- Every team member does this at the same time.

```
$ git branch
$ git switch dev
$ nano MyFile
$ git add
$ git commit -m "I made my file."
$ git push
```

Merge Conflict !!

Resolving a merge conflict

- Git will tell you which file(s) to edit.
 - The file will clearly show the conflict:

```
<<<<<<< HEAD:MyFile.txt  
This was your text.  
=====  
This is their text.  
>>>>>>>
```

6. Lab: Local Docker container

Dockerfile

- The JuiceShop repository contains a config file.
 - "*Dockerfile*" is used to define an image build.
 - Each line is an instruction to Docker.
- Read the "*Dockerfile*".
 - See if you recognize what we did yesterday.

Assignment: build container

- Let's make a container image with the full web app.
 - Like before, "*npm install*" takes a long time.
 - It will take 3 to 15 minutes.

```
$ docker build -t team1:dev .
```

Assignment: run container

- You can use the container to run your app!

```
$ docker run --rm -p 3000:3000 team1:dev
```

- Access it on <http://localhost:3000>

Assignment: run container

- Either use your host OS' browser.
 - Or test with *curl* on the DEV VM.



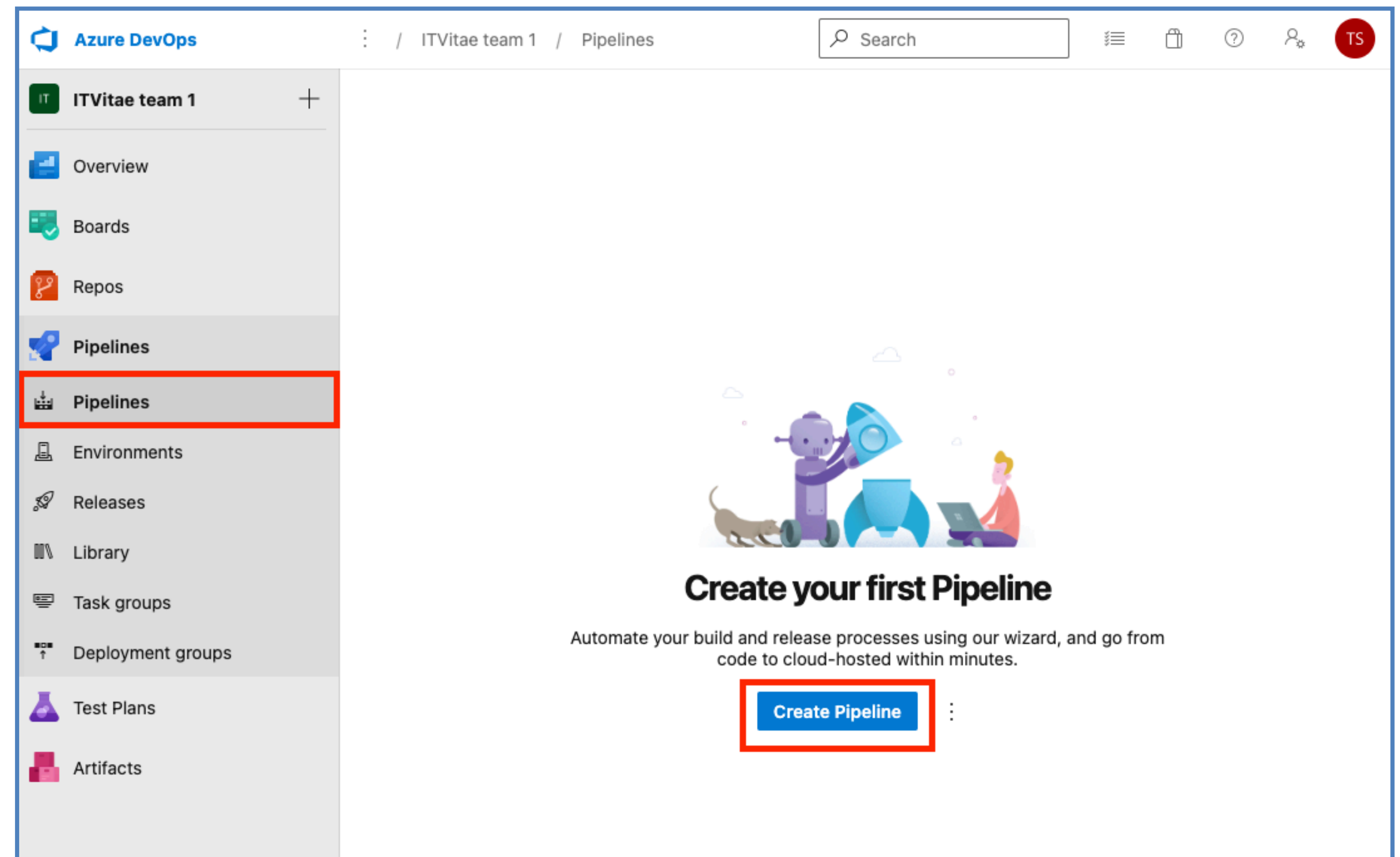
9. Lab: CI/CD pipelines

What we will make

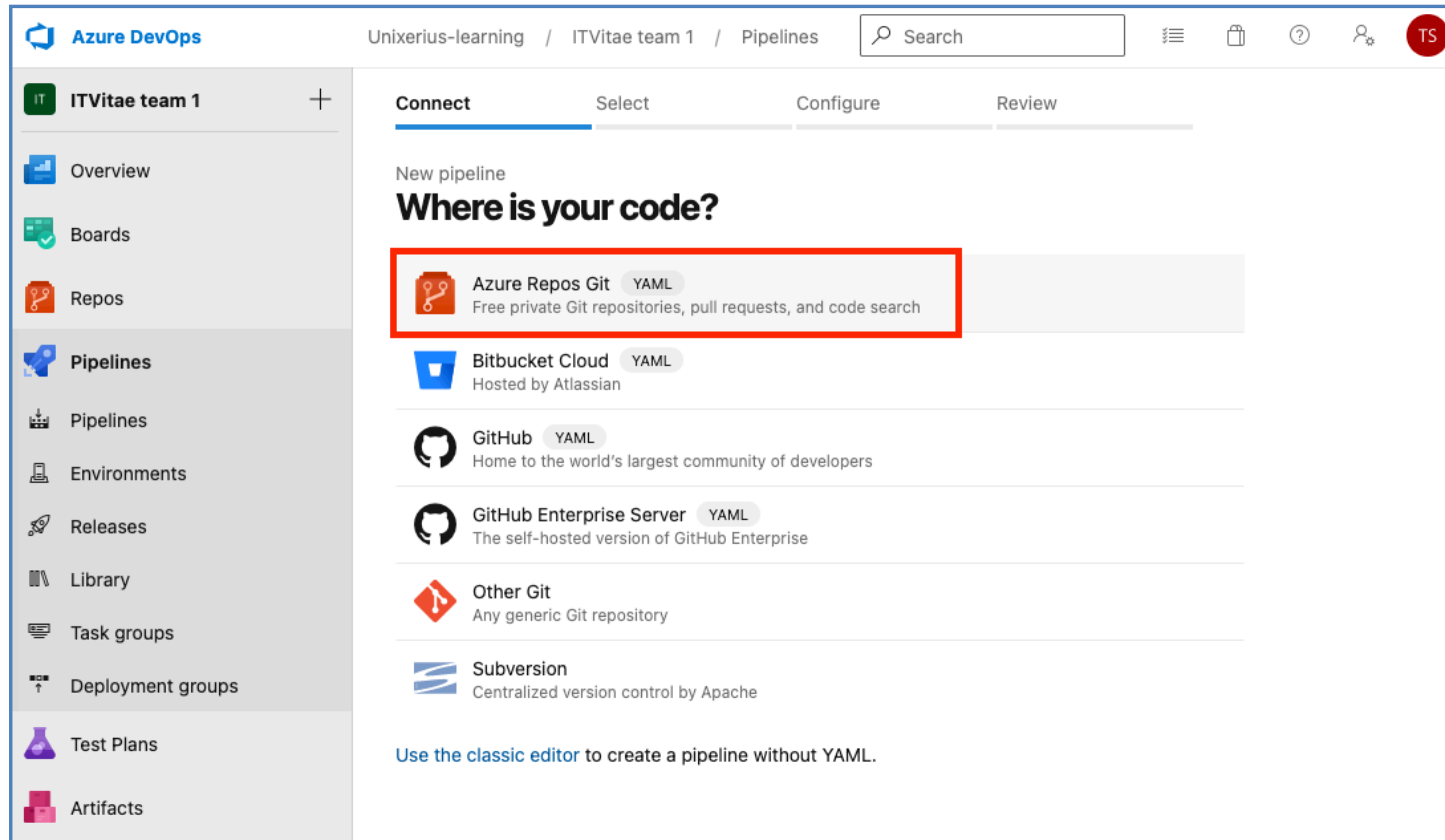
- A CICD pipeline which:
 - Logs in to Azure Container Registry.
 - Builds and pushes the JuiceShop image.
 - Logs out from ACR.
 - Orders Azure WebApp to deploy the image.

Start with a "dummy"

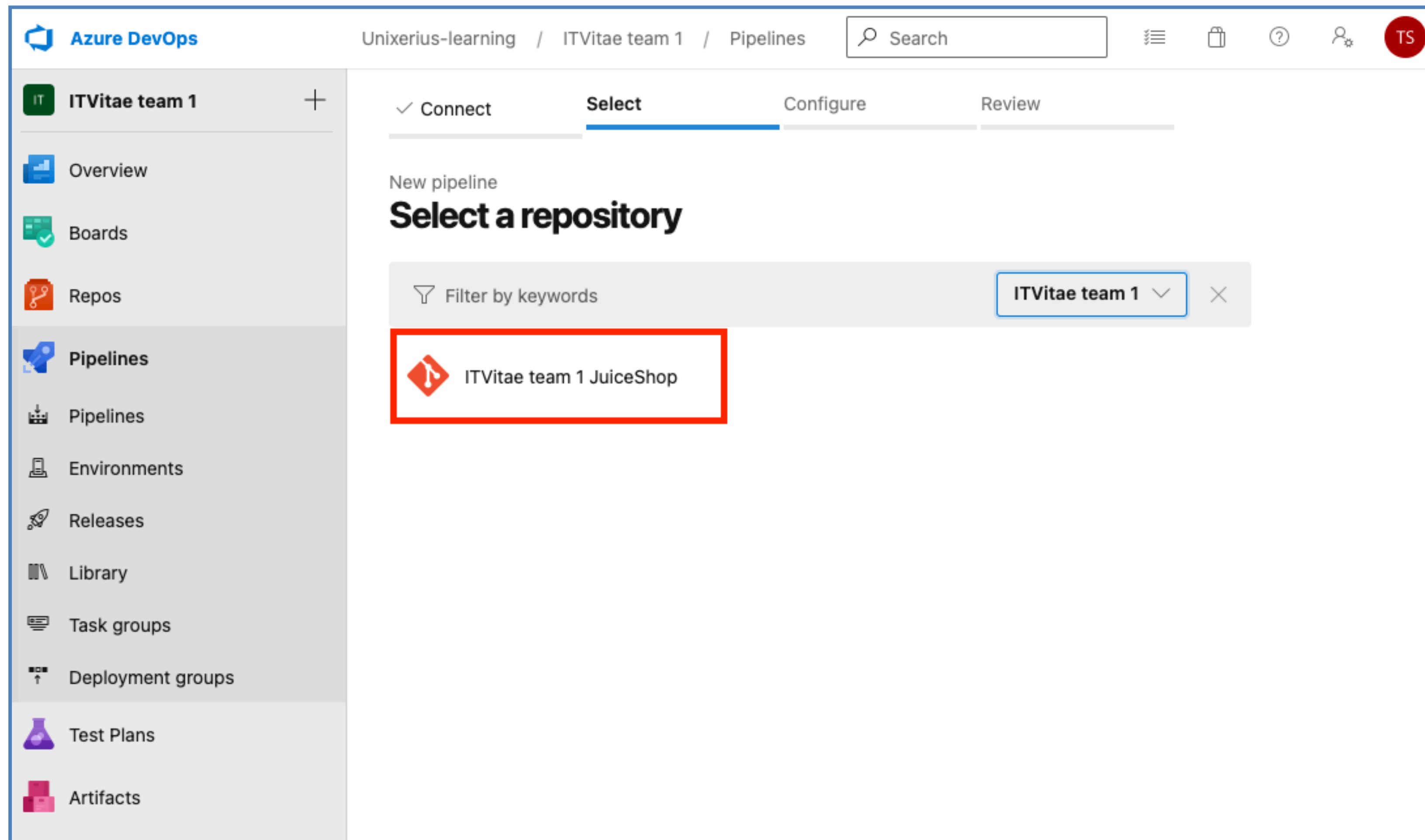
- Go to **Pipelines**.
- Create a new pipeline.



Start with a "dummy"

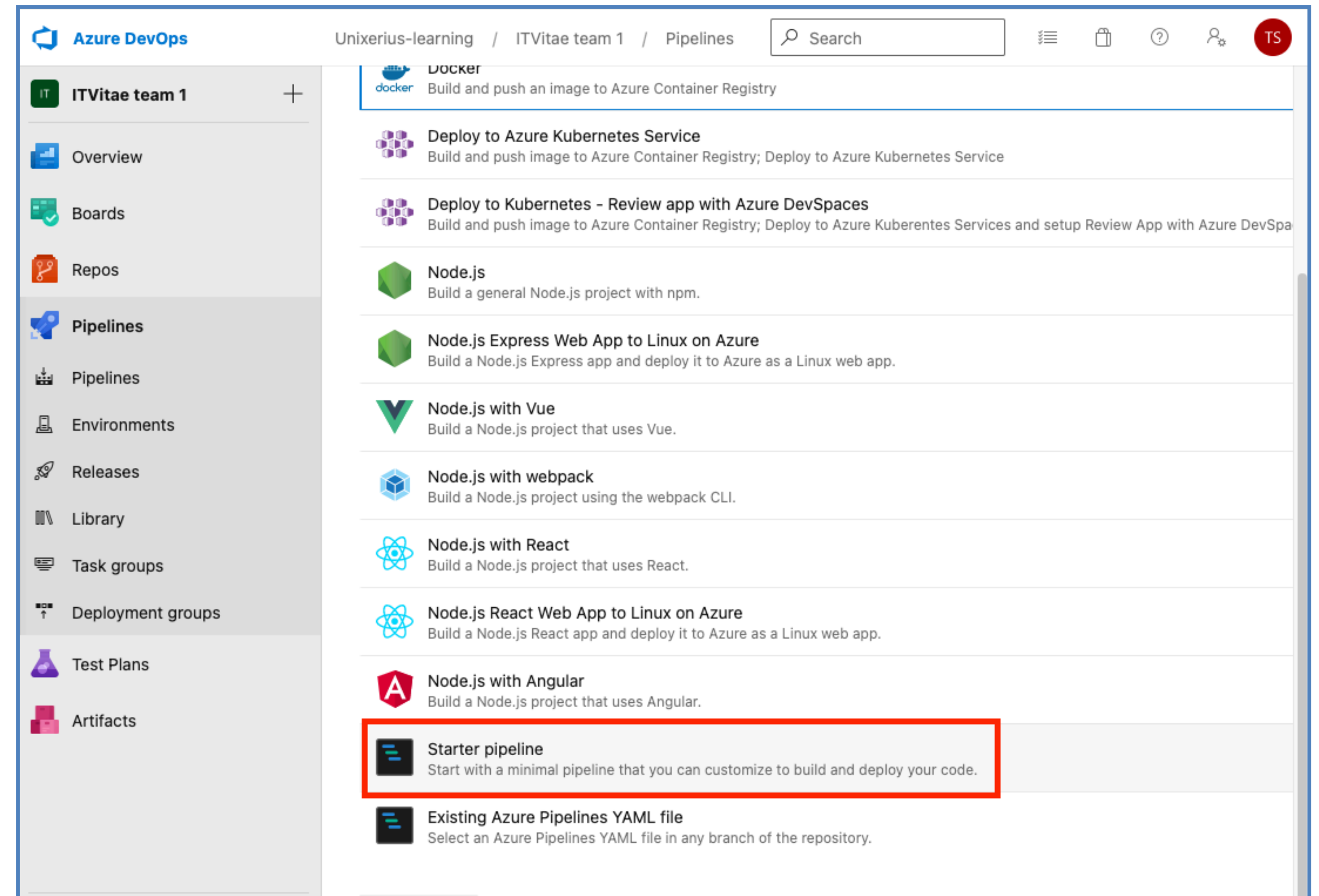


Start with a "dummy"



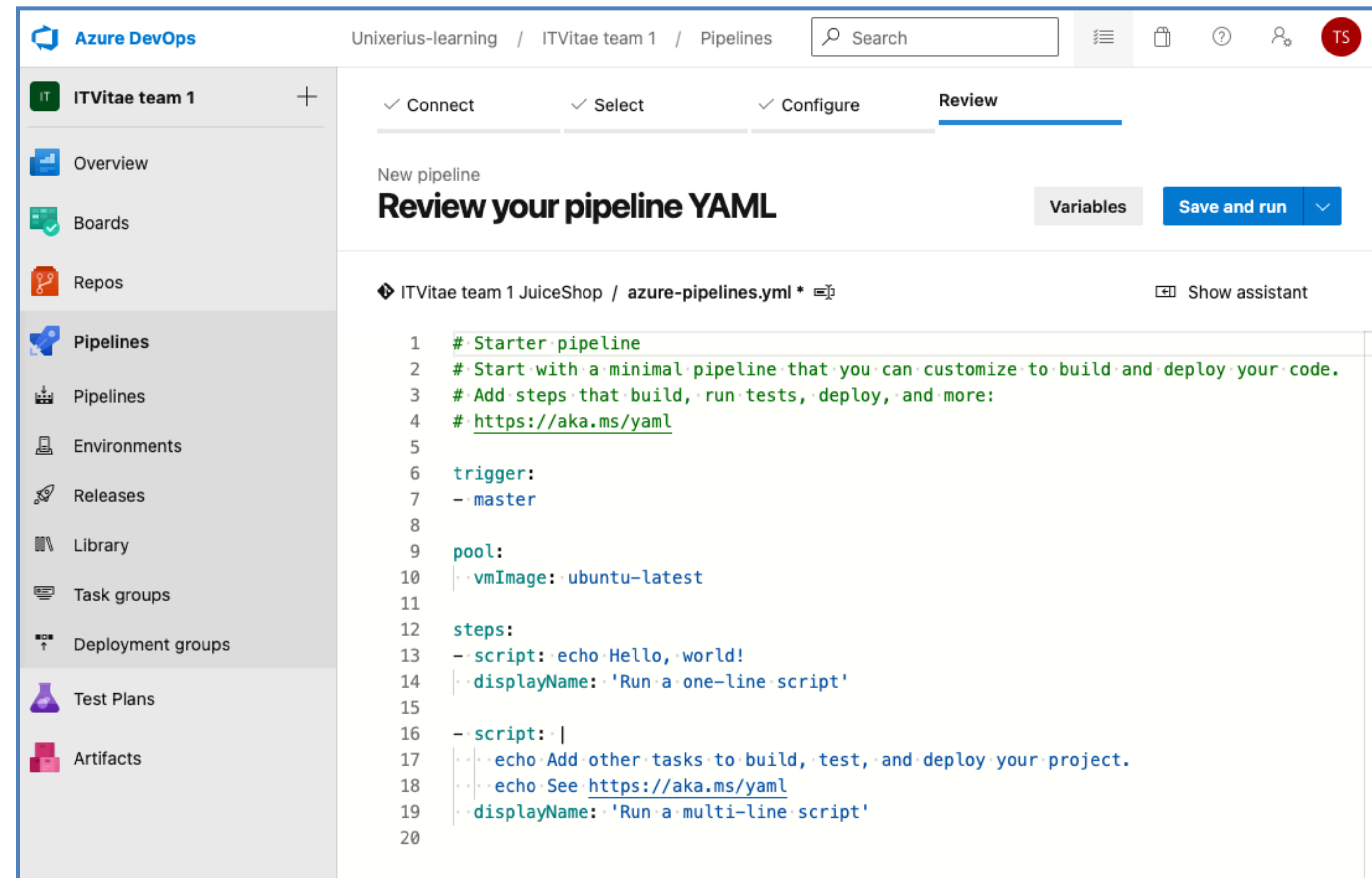
Start with a "dummy"

- Choose type:
 - "Starter pipeline".



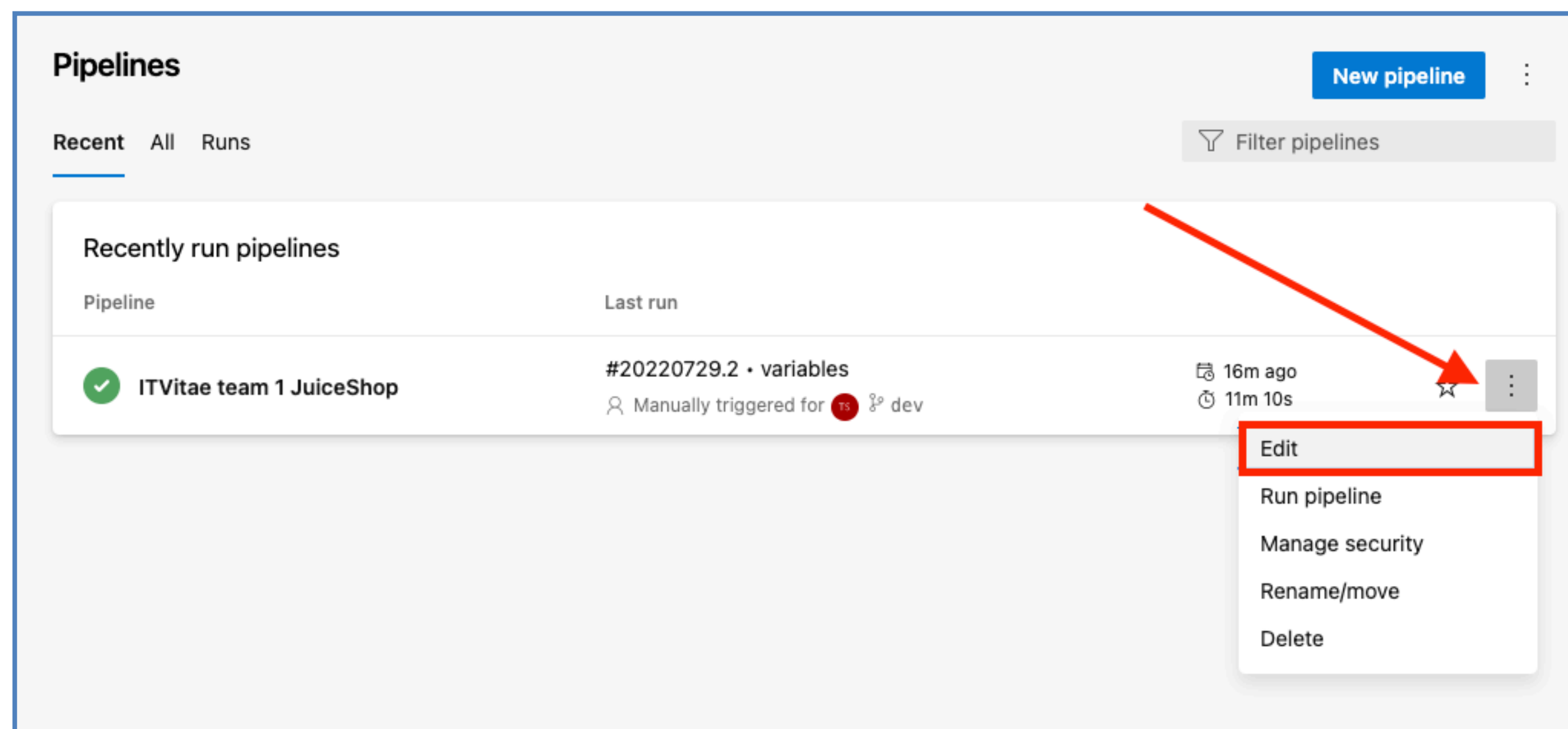
Start with a "dummy"

- A dummy is made.
- Click "Save and run".

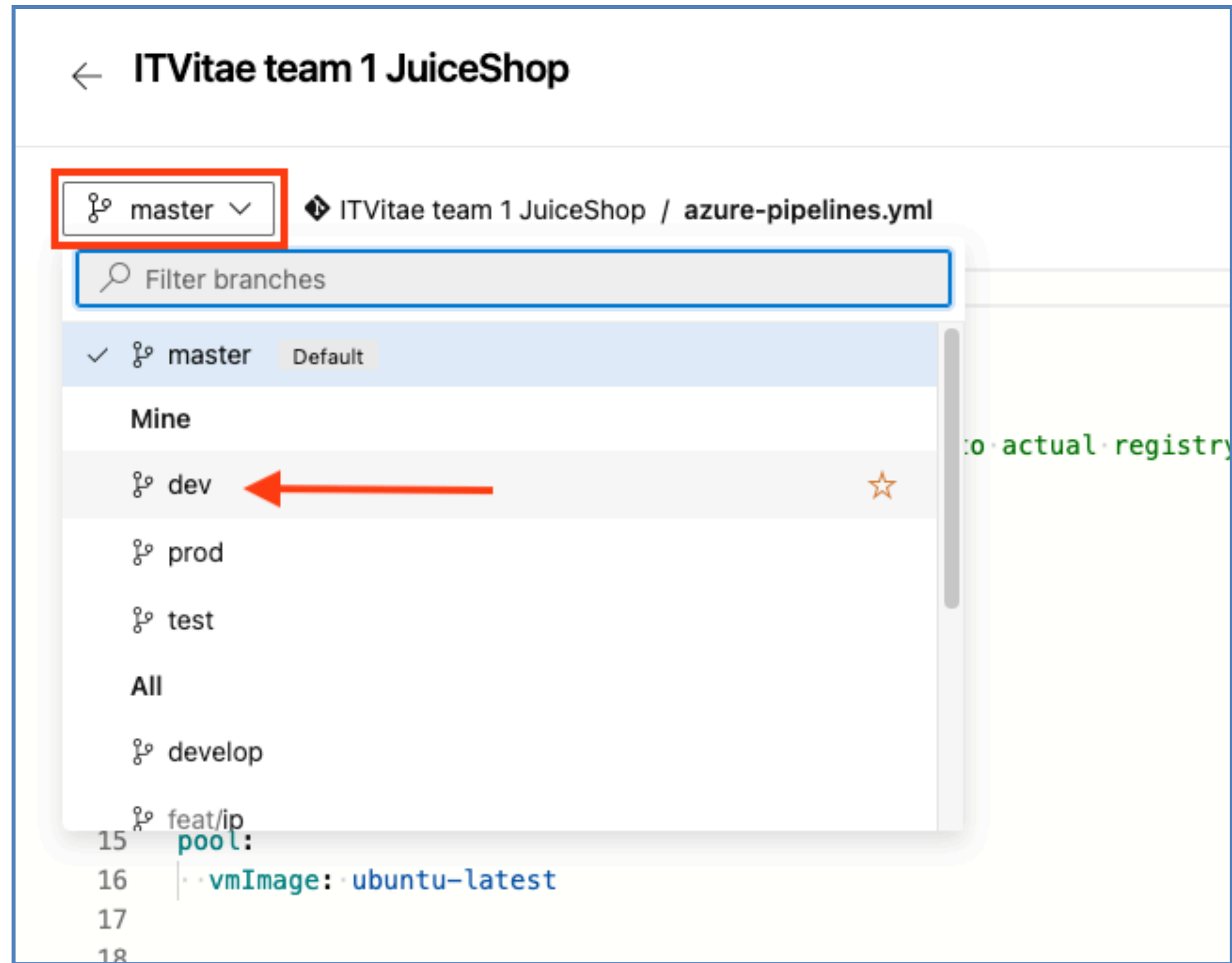


Building our pipeline

- Go back to the **Pipelines** tab.
- Edit the pipeline and choose the "dev" branch!



Building our pipeline



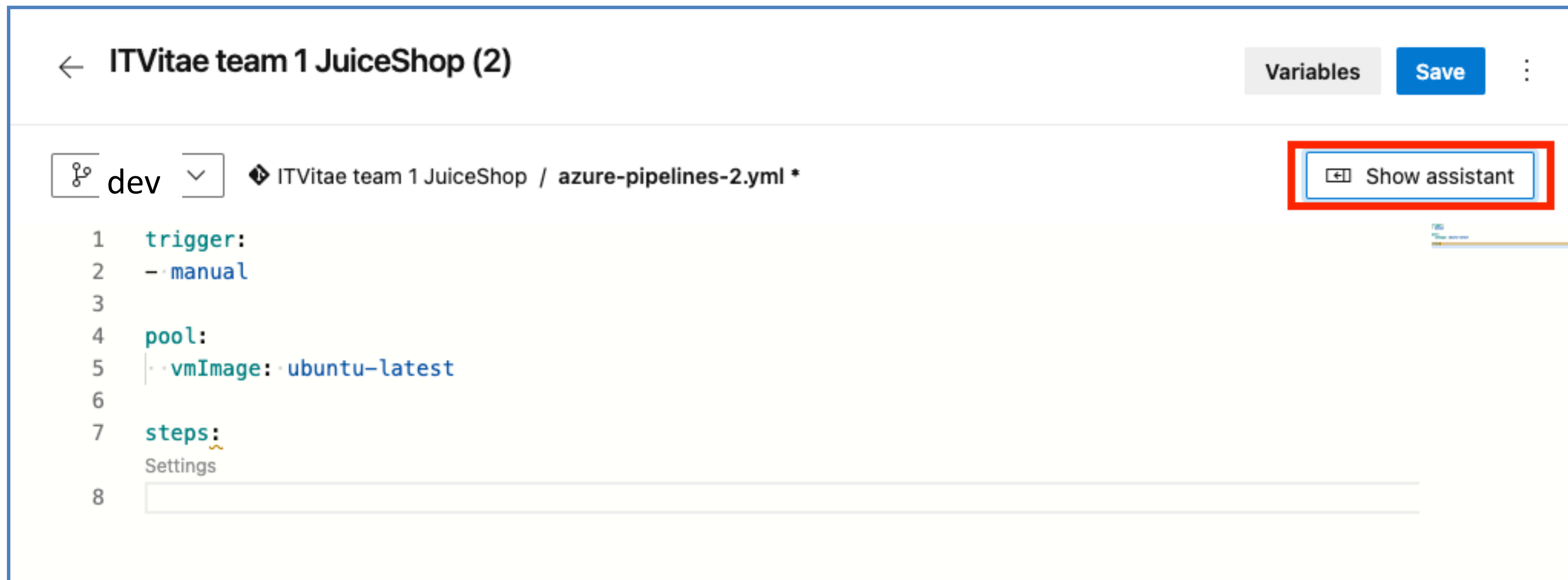
Cleanup ...

- Change the "trigger", to "*manual*".
- We can remove the dummy lines under "steps:"

```
1  trigger:
2    - manual
3
4  pool:
5    - vmImage: ubuntu-latest
6
7  steps:
8    |
```

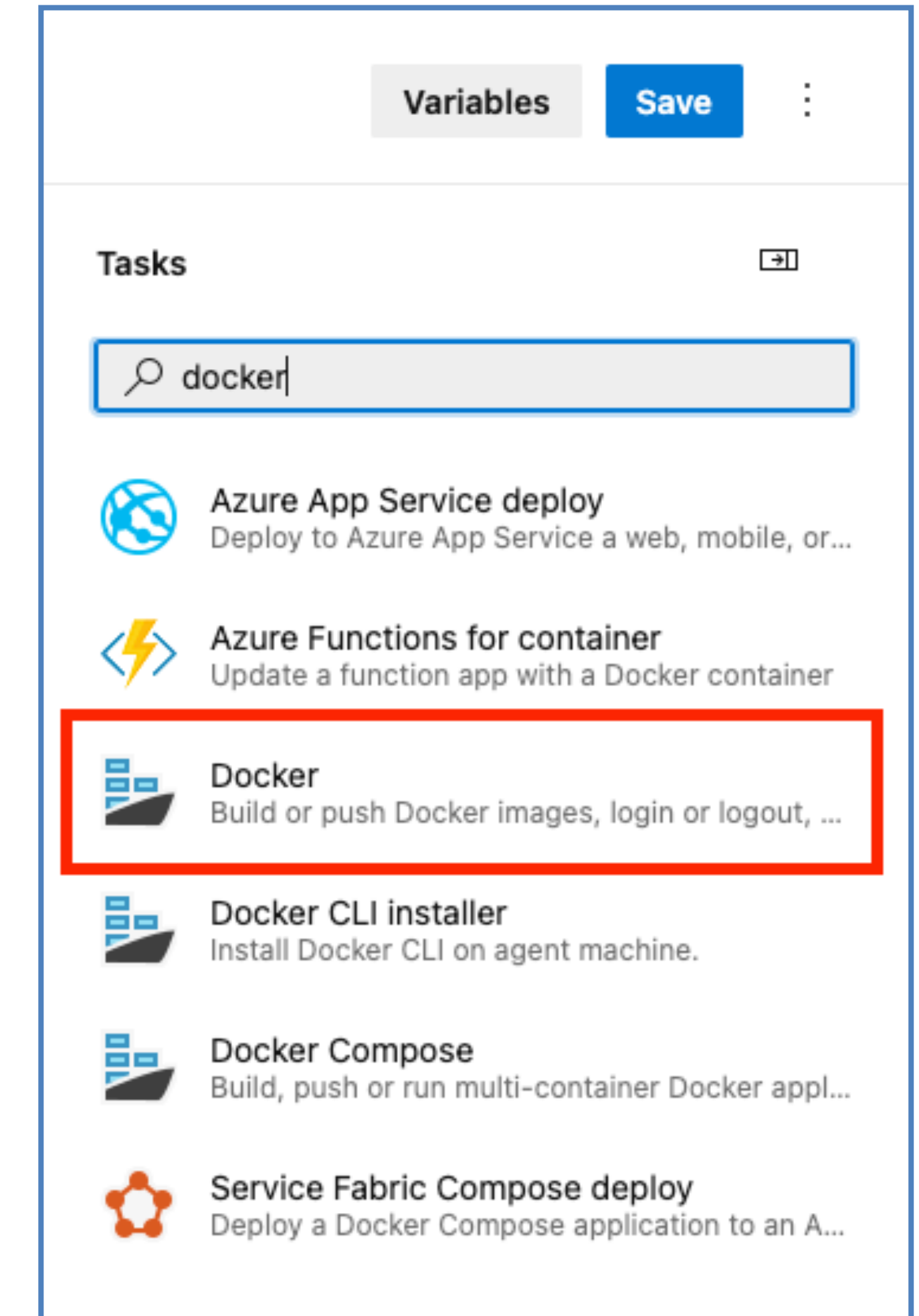
Docker login / logout

- Pull up the wizard, which has all plugins.



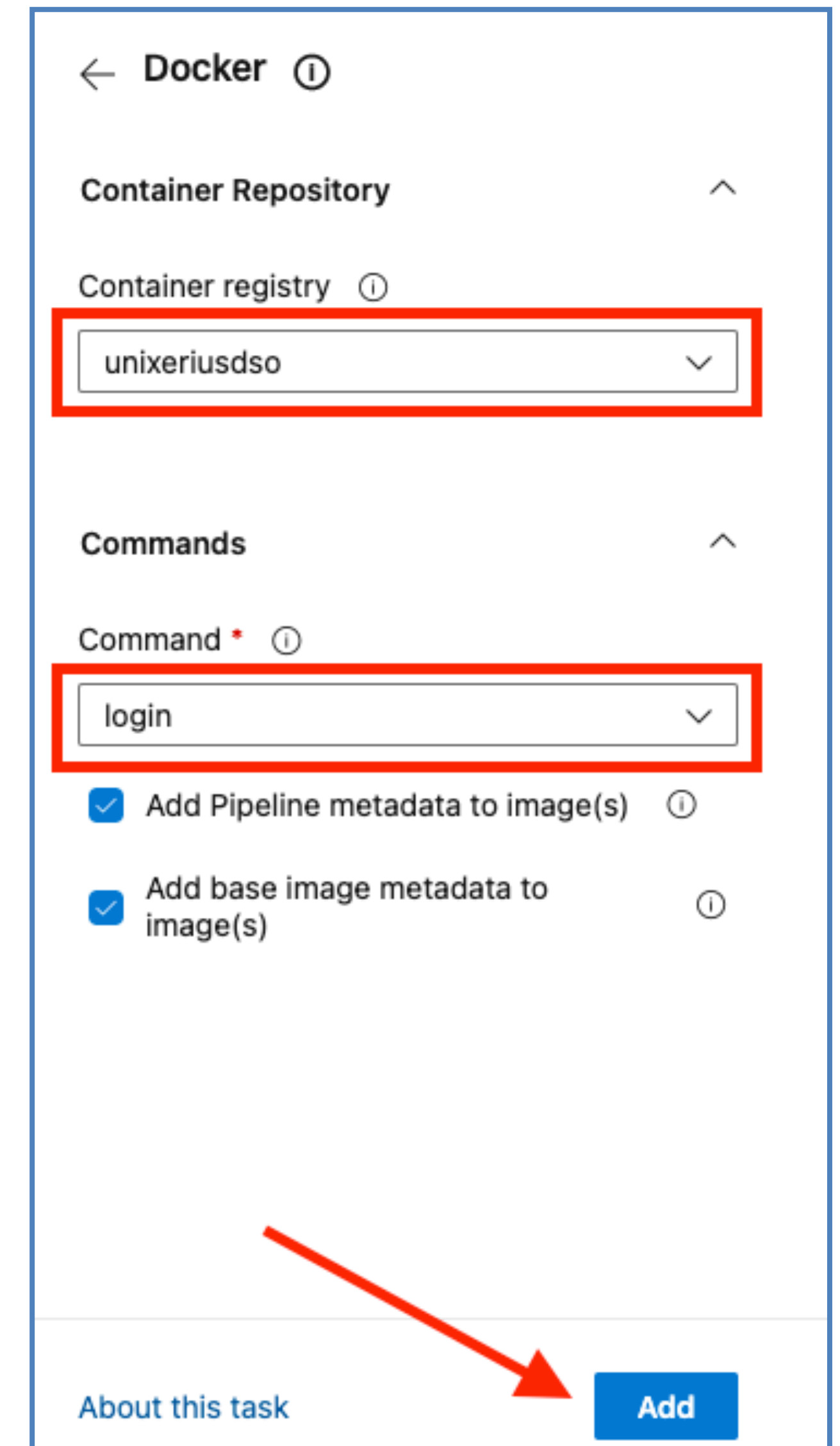
Docker login / logout

- Search for Docker,
 - Grab the one for login/logout



Docker login / logout

- Choose "*Login*" as command.
- Choose the "*unixeriudso*" registry.
- Repeat for "*Logout*".
 - Example on next slide.



The screenshot shows the Docker configuration interface for a task. It has a blue header bar with a back arrow, the text "Docker", and an information icon. Below the header, there are two expandable sections: "Container Repository" and "Commands". The "Container Repository" section is expanded, showing a dropdown menu with "unixeriudso" selected. The "Commands" section is also expanded, showing a dropdown menu with "login" selected. Below these sections, there are two checkboxes: "Add Pipeline metadata to image(s)" and "Add base image metadata to image(s)", both of which are checked. At the bottom of the interface, there is a blue button labeled "Add" and a link labeled "About this task". A red arrow points from the "Add" button towards the bottom right corner of the interface.

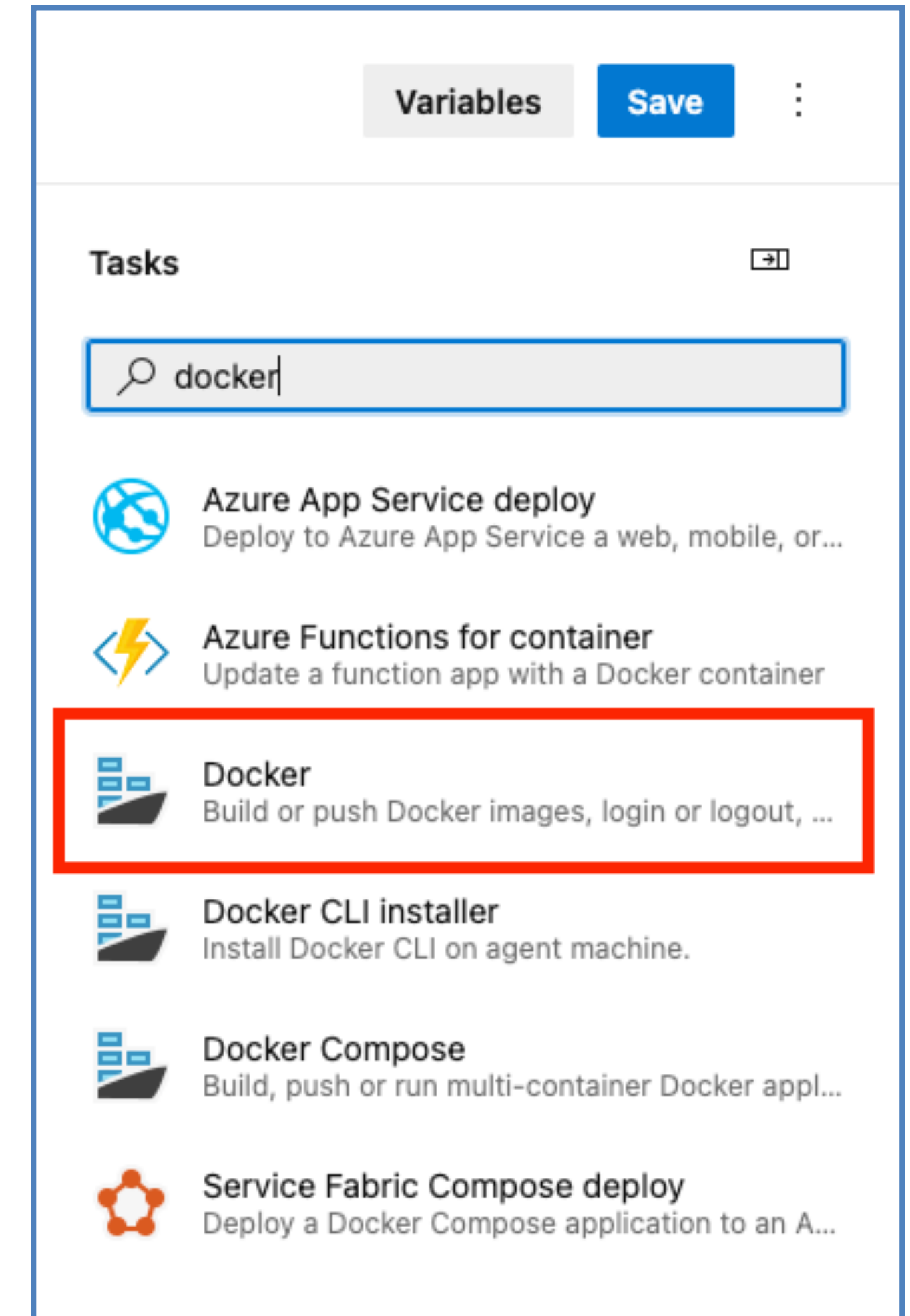
Docker login / logout

- Your code should be ->
- Validate and save.
 - Then run on Dev branch.

```
1  trigger:
2  - manual
3
4  pool:
5  | vmImage: ubuntu-latest
6
7  steps:
8  Settings
9  - task: Docker@2
10 | inputs:
11 |   containerRegistry: 'unixeriusdso'
12 |   command: 'login'
13 Settings
14 - task: Docker@2
15 | inputs:
16 |   containerRegistry: 'unixeriusdso'
17 |   command: 'logout'
```

Docker build and push

- If the pipeline ran OK,
 - It's time to build the image.
- Go to the editor again,
 - Make sure you're on "dev"!
 - Open the wizard again.
 - Select the same Docker task.



Docker build and push

- Chose "*BuildAndPush*" as command.
- Chose the "*unixeriusdso*" registry.
- Set repository to "team1",
 - Adjust for your team!
- Set tags to "dev",
- Click "add".

Container Repository ^

Container registry ⓘ

UnixeriusDSO v

Container repository ⓘ

team1

Commands ^

Command * ⓘ

buildAndPush v

Dockerfile * ⓘ

**/Dockerfile

Build context ⓘ

**

Tags ⓘ

dev

☒ Add Pipeline metadata to image(s) ⓘ

Docker build + push

- Your code should be ->
- Save and run.
 - Building takes 8-10 mins.

```
1 trigger:
2   - manual
3
4 pool:
5   - vmImage: ubuntu-latest
6
7 steps:
8   Settings
9   - task: Docker@2
10    inputs:
11      containerRegistry: 'UnixeriusDS0'
12      command: 'login'
13    Settings
14    - task: Docker@2
15      inputs:
16        containerRegistry: 'UnixeriusDS0'
17        repository: 'team3'
18        command: 'buildAndPush'
19        Dockerfile: '**/Dockerfile'
20        tags: 'dev'
21      Settings
22      - task: Docker@2
23        inputs:
24          containerRegistry: 'UnixeriusDS0'
25          command: 'logout'
```

Run on Azure WebApp

- Your team already has a WebApp!
 - Adjust for the right team name.

<http://unixeriussdo-team1.azurewebsites.net/>

Run on Azure WebApp

- Go back to the pipeline editor.
 - Make sure you're on the "dev" branch!
- At the bottom, we will add a block of code.
 - This tells Azure WebApp to load your container.

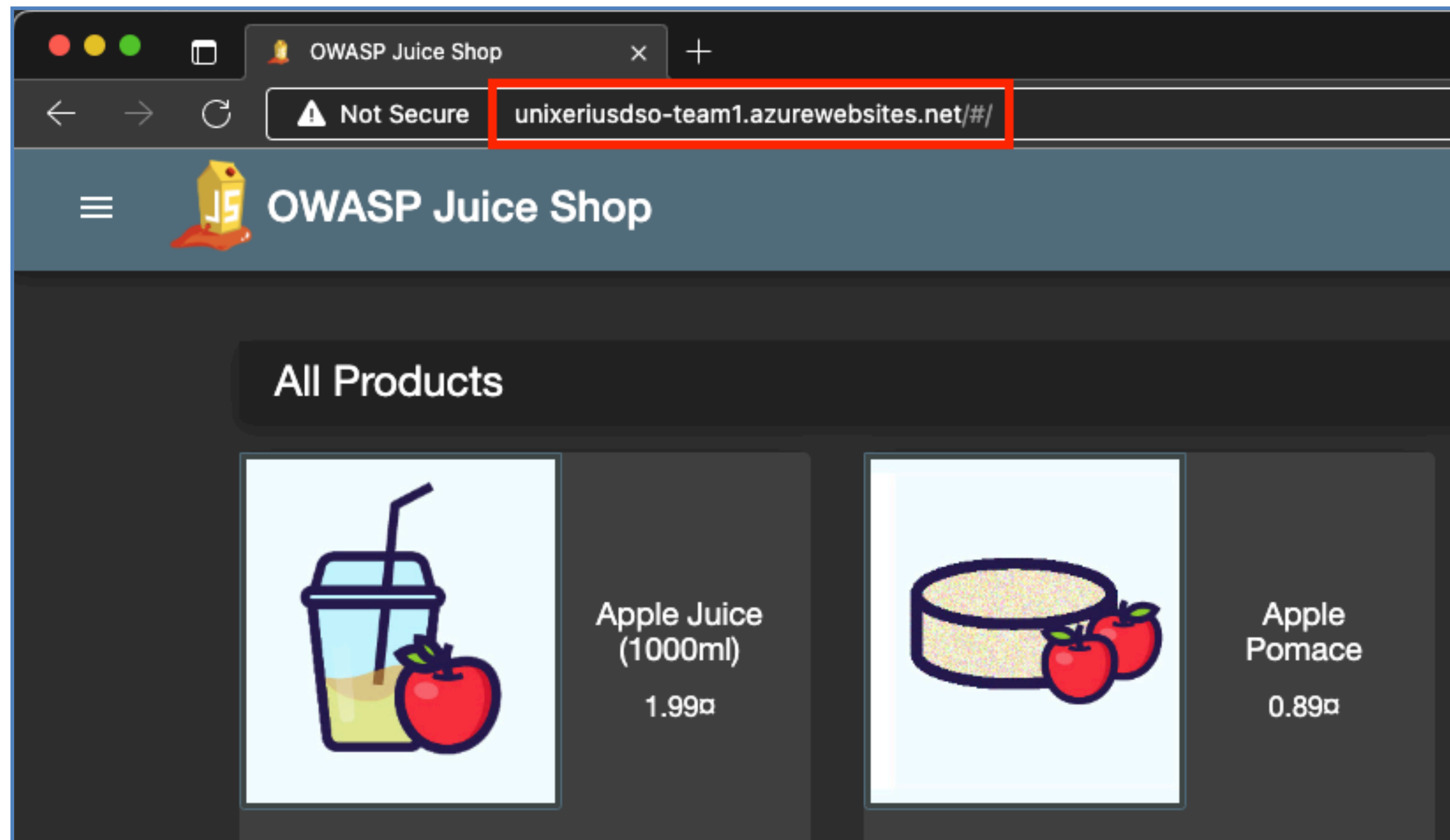
The code

- Is also available as "[pipeline-step1-build-run.yml](#)".
 - The dashes shown below are a single dash!
 - `task: AzureWebAppContainer@1`
`inputs:`
 - `azureSubscription: 'Azure Unixerius Learning'`
 - `appName: 'unixeriusdso-team1'`
 - `containers: 'unixeriusdso.azurecr.io/team1:dev'`
 - `appSettings: '-Port 3000'`
 - `configurationStrings: '-acrUseManagedIdentityCreds true'`

Deployment to Dev!

- If you run the pipeline again, it will:
 - Build and push your image to ACR.
 - Run your container as WebApp.
- After deployment, it will take a few minutes.
 - Our WebApp instances are a bit slow. 🤔

Deployment to Dev!



Checkpoint!

- Does everyone have:
 - A pipeline on the "dev" branch.
 - Which builds and pushes to ACR?
 - Which runs the container?
- Have you tested this?

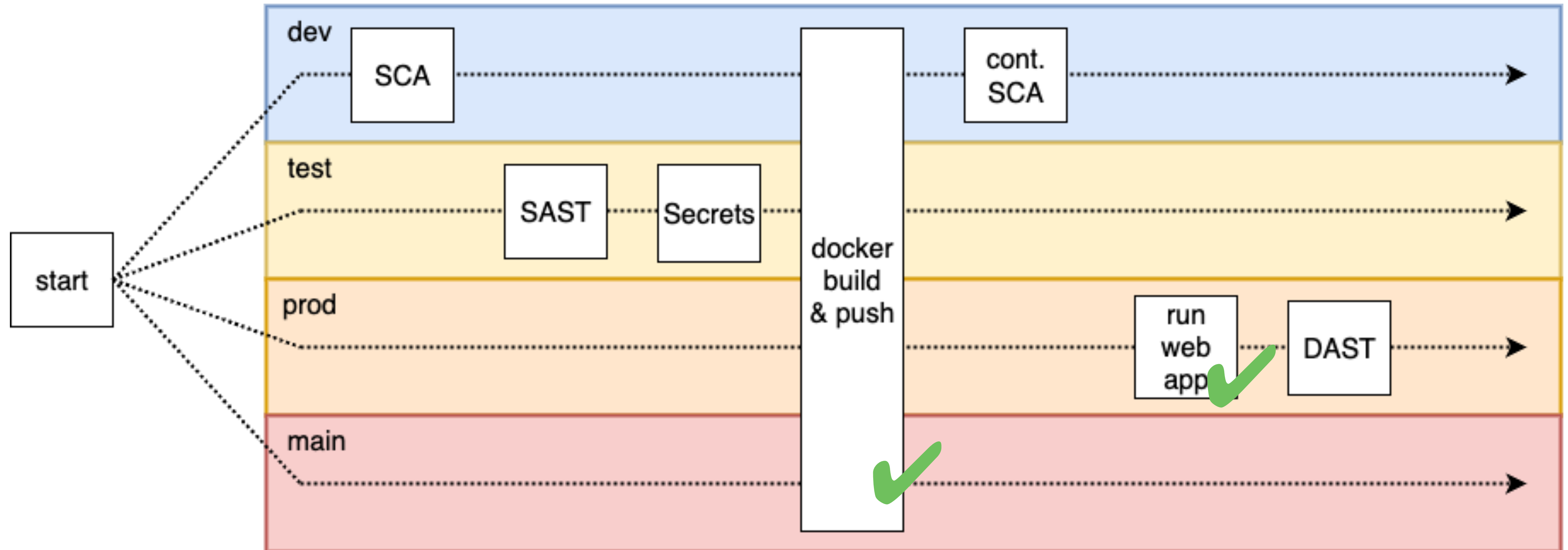


10. Lab: Quick cleanup

I have a spoiler file.

- Available as "[pipeline-step1-build-run.yml](#)".
 - It prepares you for "stages" and more.
- Use it for your "dev" pipeline.
 - Replace the "team1" with your team.

Our final pipeline goal



Checkpoint!

- Does everyone have:
 - My spoilers as their pipeline?
 - Does it still build and deploy?
- Have you tested this?



Closing

What have we achieved?

- We practiced with Git.
- We ran JuiceShop in Docker, locally.
- We made our first pipeline.
- We ran JuiceShop on Azure, via CI/CD.

Tomorrow

- We will automate functional testing.
- We will look into software vulnerabilities,
 - DevSecOps fundamental concepts, and
 - Threat modelling.

Relevant reading

Topic	Book
Vulnerability management	Ch 6
Threat assessments	Ch 8
Agile software (and security) testing	Ch 11

Reference materials

Resources

- [Understanding the TCO of "serverless"](#)
- [ArgoCD for beginners](#)
- In-depth: [So you think you know Git?](#)
- In-depth: [So you think you know Git? Part 2](#)