

# DevSecOps, day 2



# 2. Lab: Tasks completed

# You've done some work

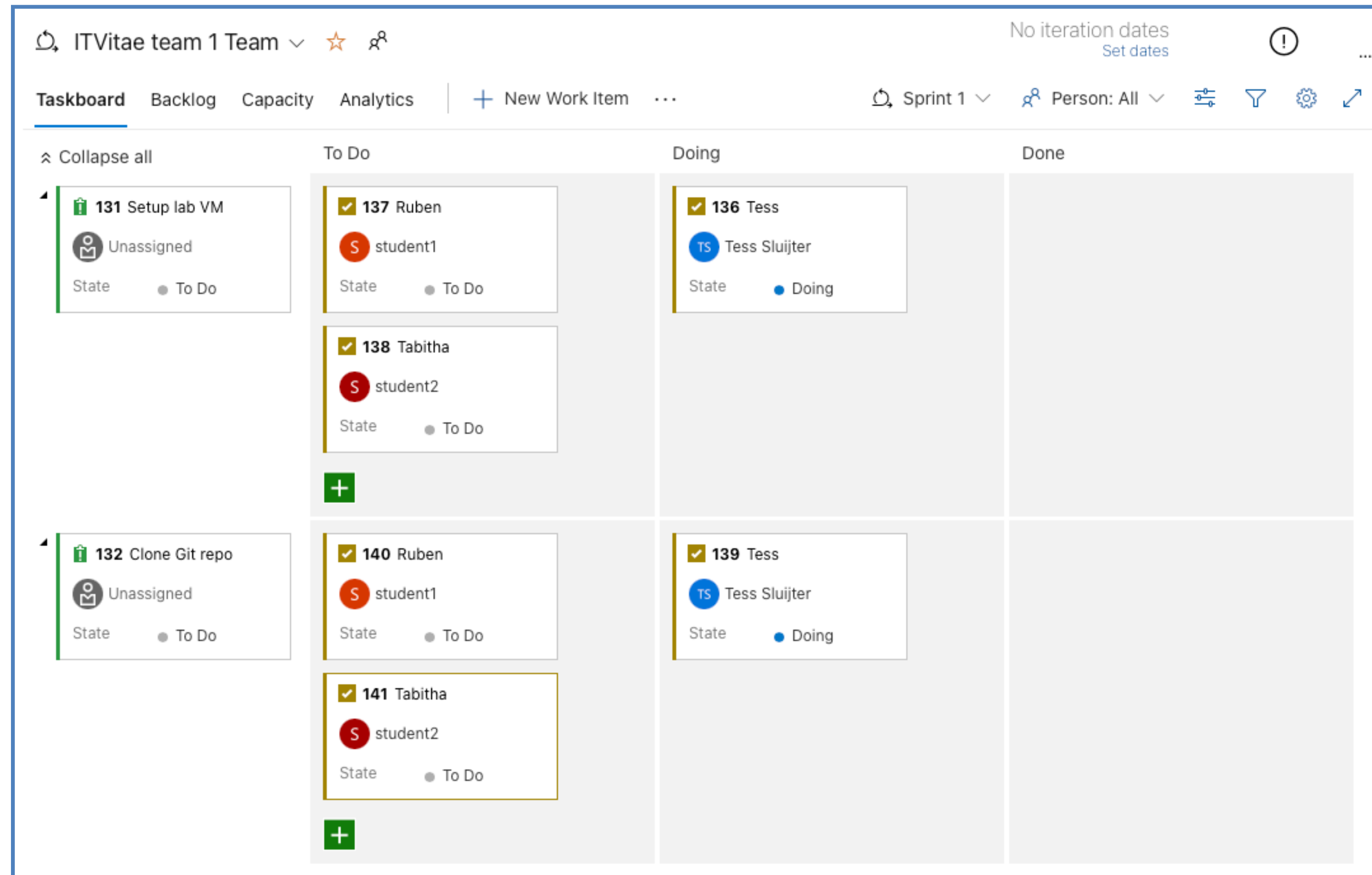
- Yesterday, you already completed a few stories!
  - You installed your Dev Workstations.
  - You cloned the Git repositories.
  - You got the local JuiceShop to run.

# 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:



# For example

- To:

The screenshot displays a Jira team board for 'ITVitae team 1 Team'. The board is set to 'No iteration dates' and shows a Kanban workflow with three columns: 'To Do', 'Doing', and 'Done'. On the left, there is a sidebar with a 'Collapse all' button and two work items, 131 and 132, both in the 'Done' state. The 'To Do' column is empty except for a green plus icon. The 'Doing' column is also empty. The 'Done' column contains three work items: 136 (Tess Sluijter), 137 (student1), and 138 (student2), all in the 'Done' state. Below these are items 139 (Tess Sluijter) and 140 (student1), also in the 'Done' state. The top navigation bar includes links to 'Taskboard', 'Backlog', 'Capacity', and 'Analytics', along with a '+ New Work Item' button. The right side of the top bar shows 'Sprint 1', 'Person: All', and various filter and settings icons.

# 4. Lab: Branches

# Making extra branches

- You can do this from your Dev Workstation.

```
$ cd ~/Team1JS  
$ git checkout master  
$ git branch dev; git push --all
```

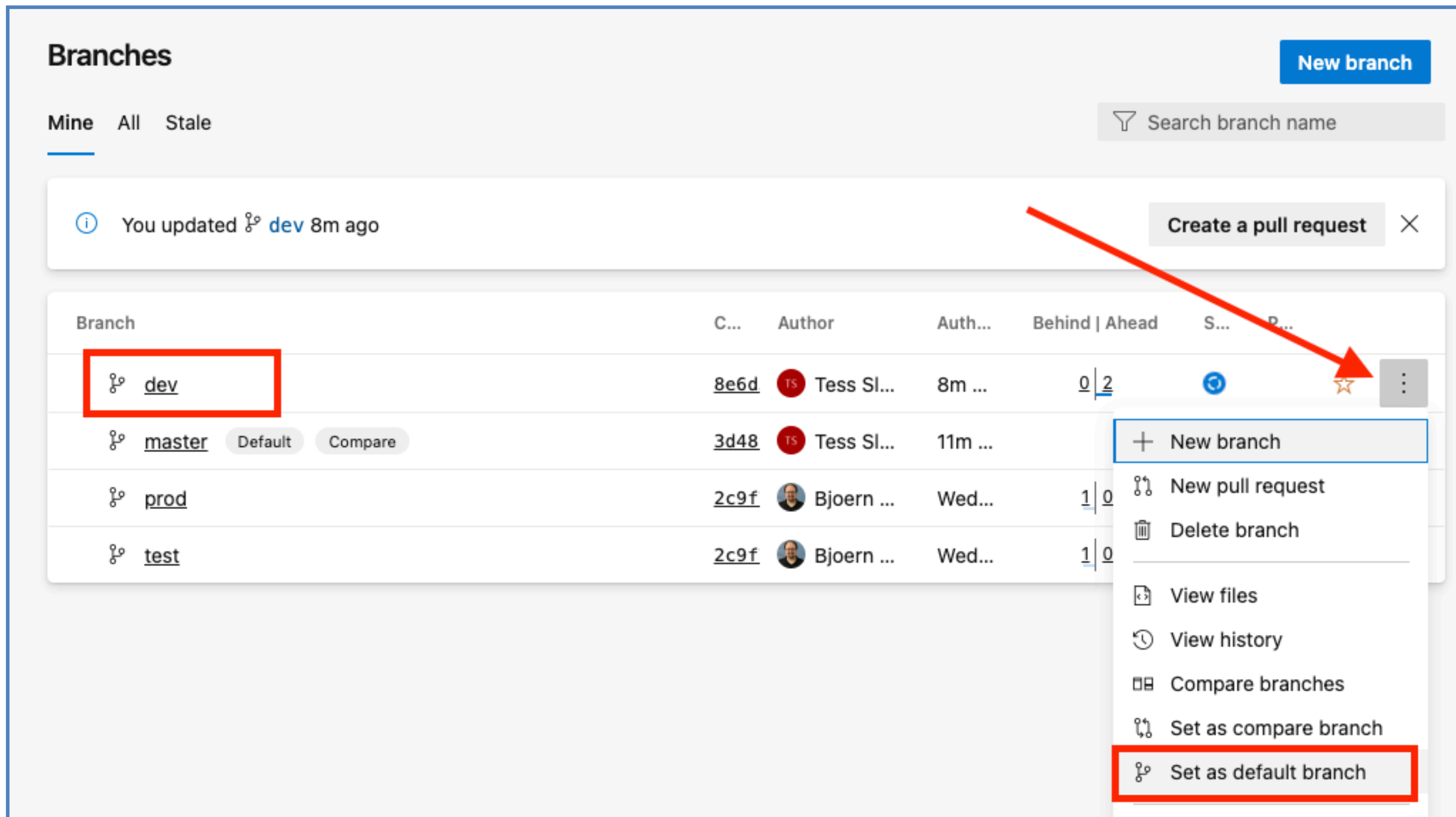
- Or in the Azure Repositories GUI.



# Making extra branches

- Make three new branches, all FROM master:
  - dev
  - test
  - prod
- All four are now at the exact same state.

# 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> <span>Default</span> <span>Compare</span>	<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**

# A quick merging exercise

- One team member makes a new file,
  - On the "dev" branch. Call it "*team.txt*".
  - They put their name in the file and save it.
- Commit that new file to the "dev" branch.
- Then push to Azure Git.

# A quick merging exercise

- The other team members can now "git pull".
  - Their "dev" branch should include "team.txt".
- The other team members now add their name.
  - Each in their local copy of the file.
- Commit to "dev" branch.

# A quick merging exercise

- In a team with more than two,
  - Whomever first pushes, has it easy. ;)
  - The others will be told to first "*git pull*".
- And when they "*git pull*", they will be told to merge.
  - Follow the instructions.
  - Then "*git push*" again.

# A quick merging exercise

- This will take some messing around!
- Normally you don't all dogpile onto the same file.
  - You usually work on the files for your feature.

# Hooray for extensions!

- A pull req might lead to a merge conflict.
  - AzDO does not have a nice way to handle this.
- To make this easier, I have added an AzDO plugin:
  - [PullReq merge conflict tab](#)

# Checkpoint!

- Does everyone have:
  - Three new branches? dev/test/prod
- Did you prove they all work?!





# 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

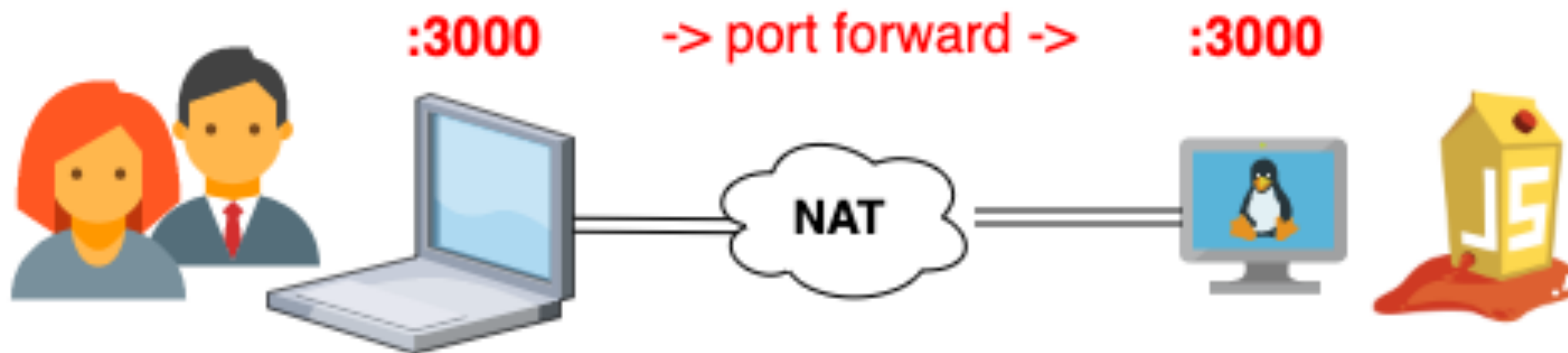
- You can use the container to run your app!

```
$ 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.



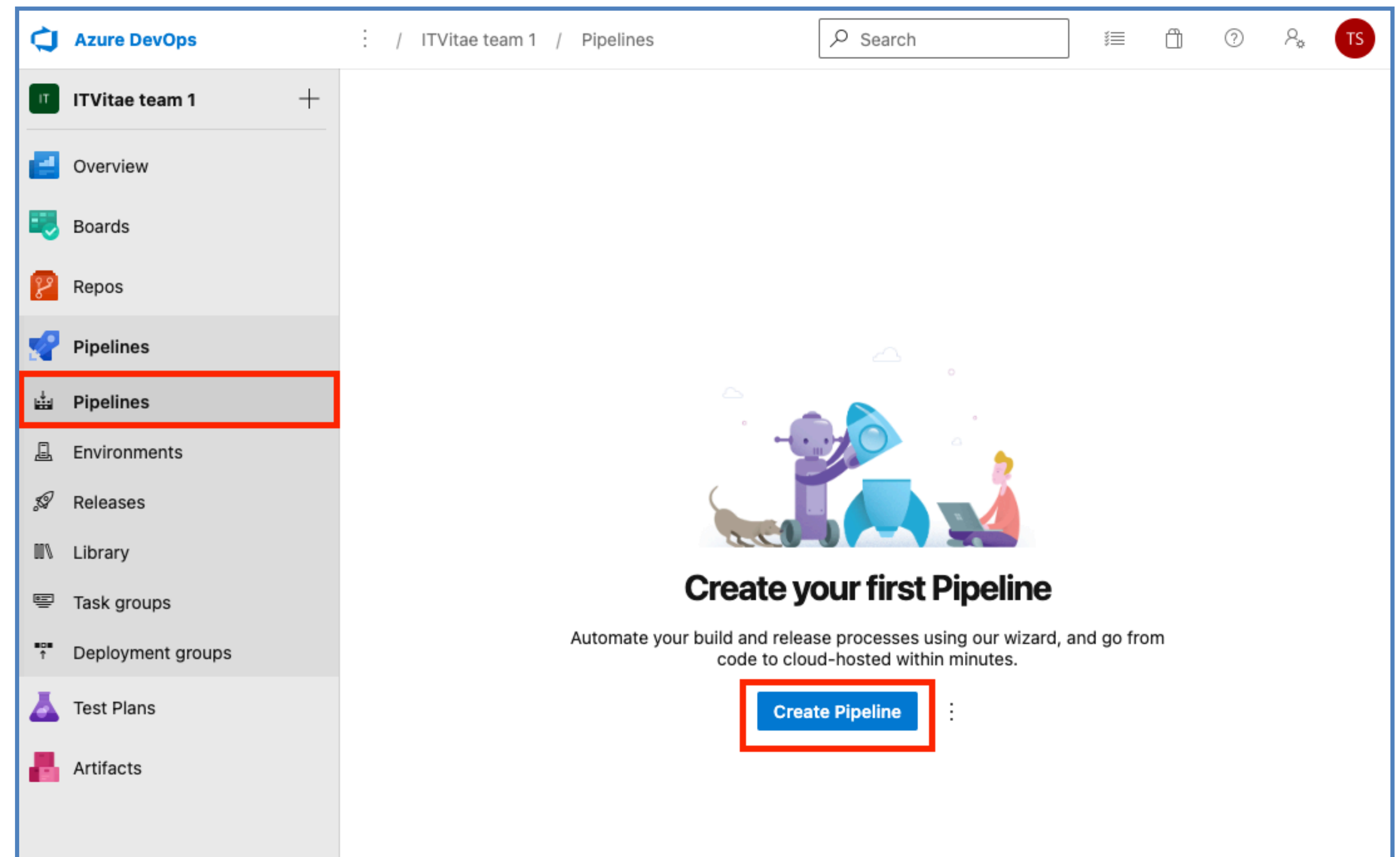
# 9. Lab: CI/CD pipelines

# What we will make

- A CI/CD 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.

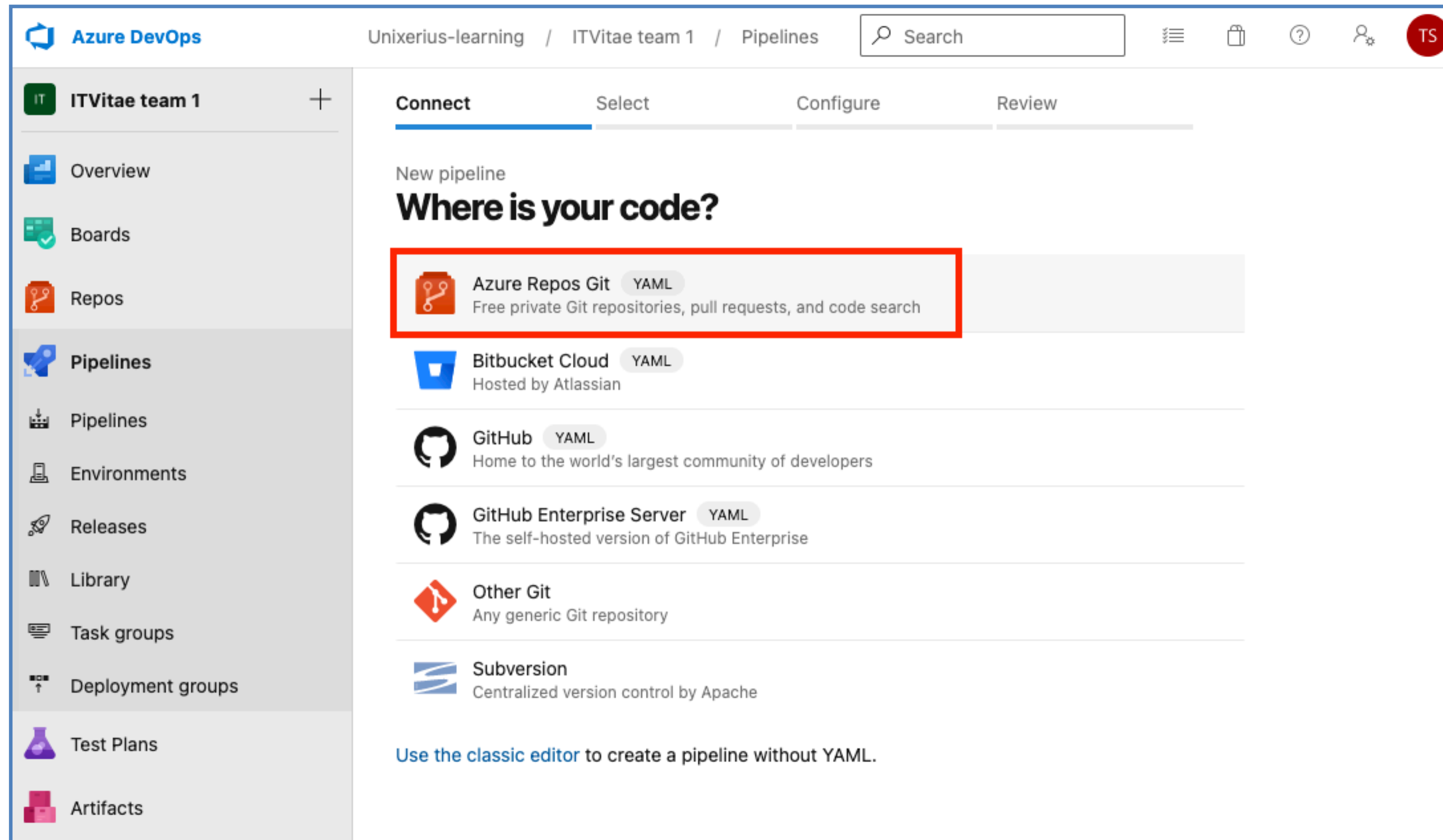
# Start with a "dummy"

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

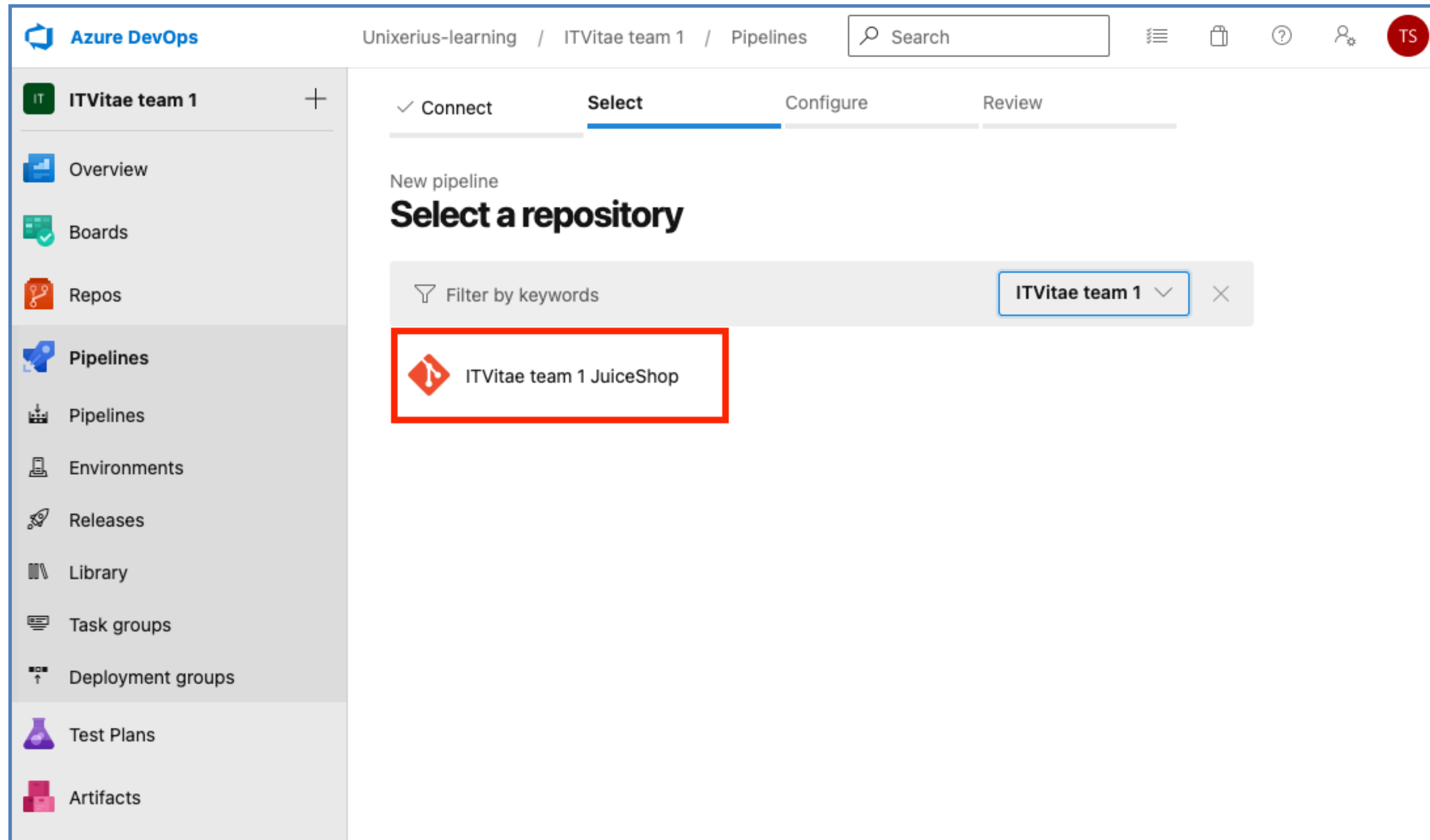




# Start with a "dummy"

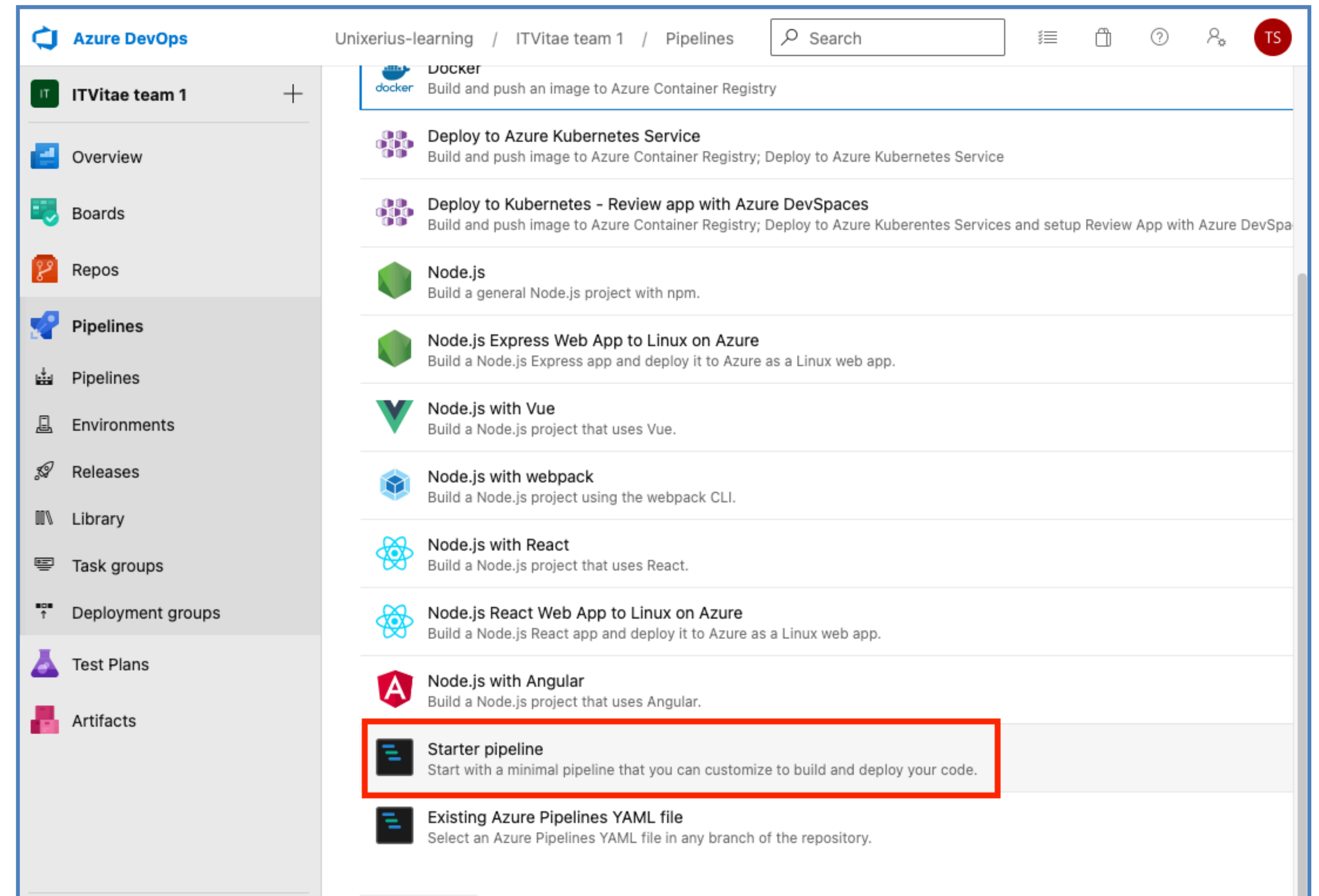


# Start with a "dummy"



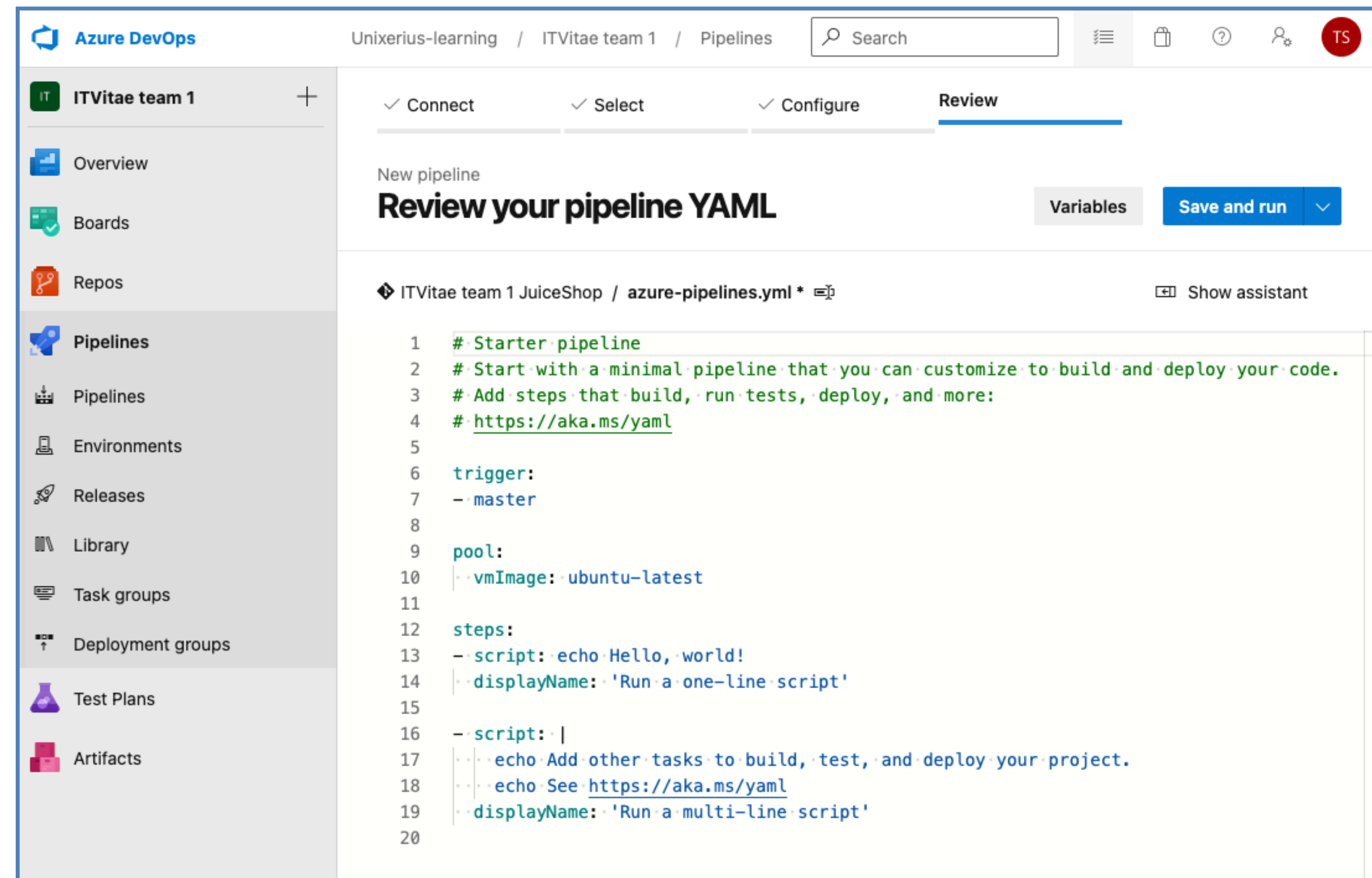
# Start with a "dummy"

- Choose type:
  - "Starter pipeline".



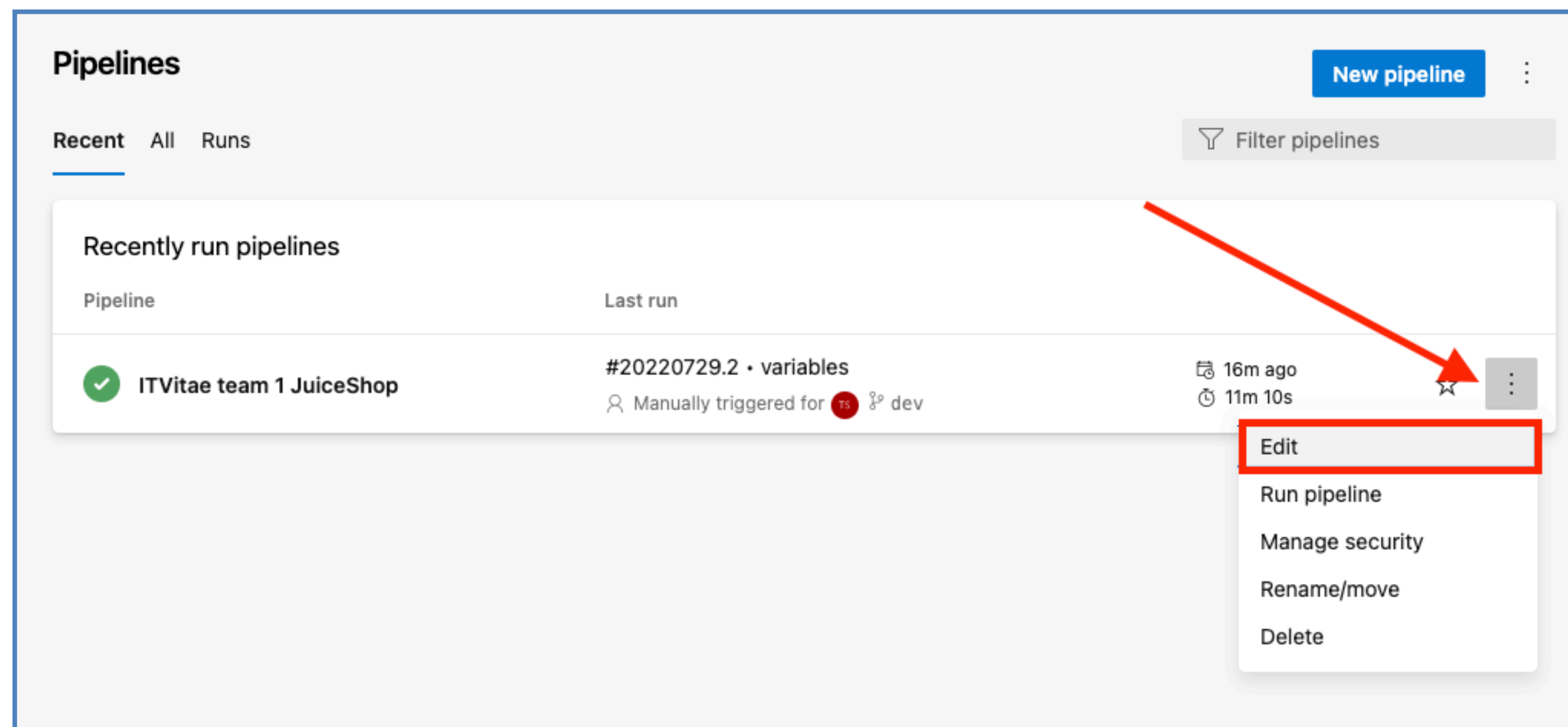
# Start with a "dummy"

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

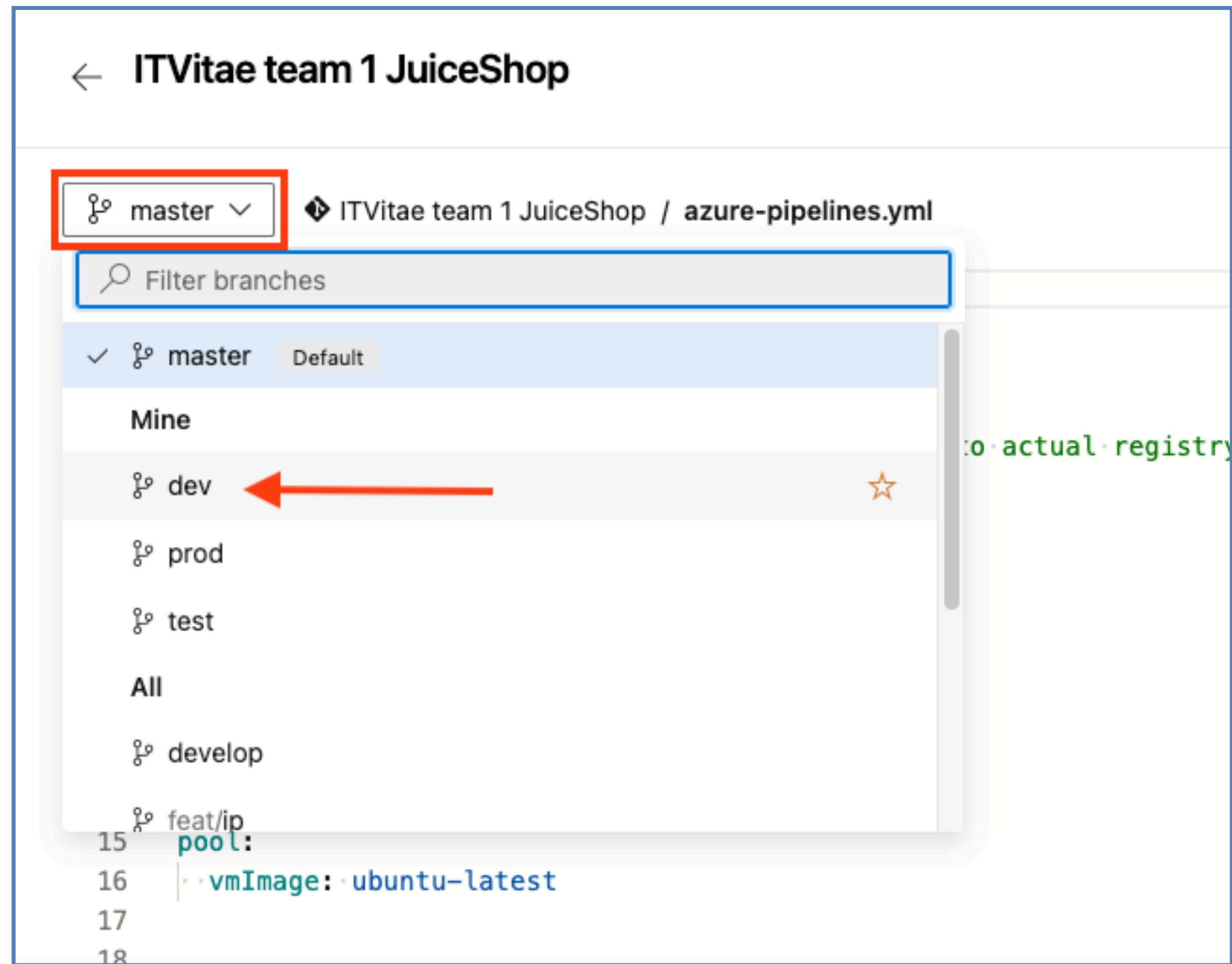


# Building our pipeline

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



# Building our pipeline





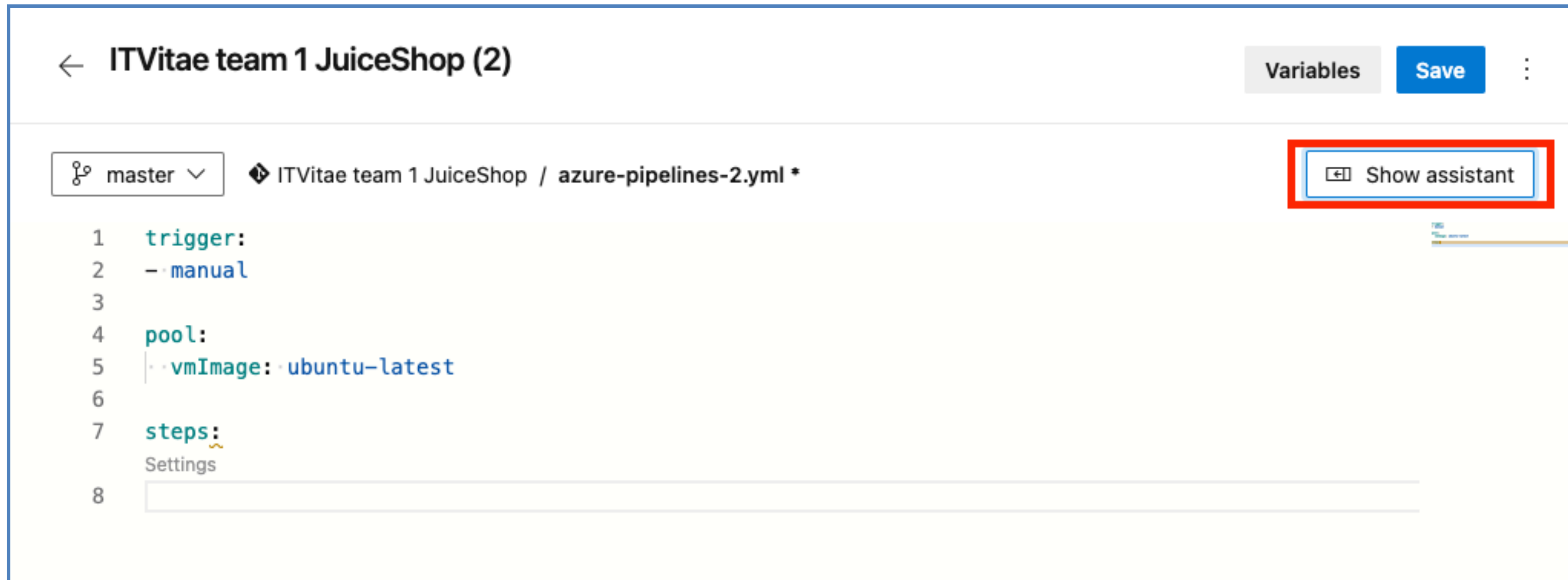
# Cleanup ...

- 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

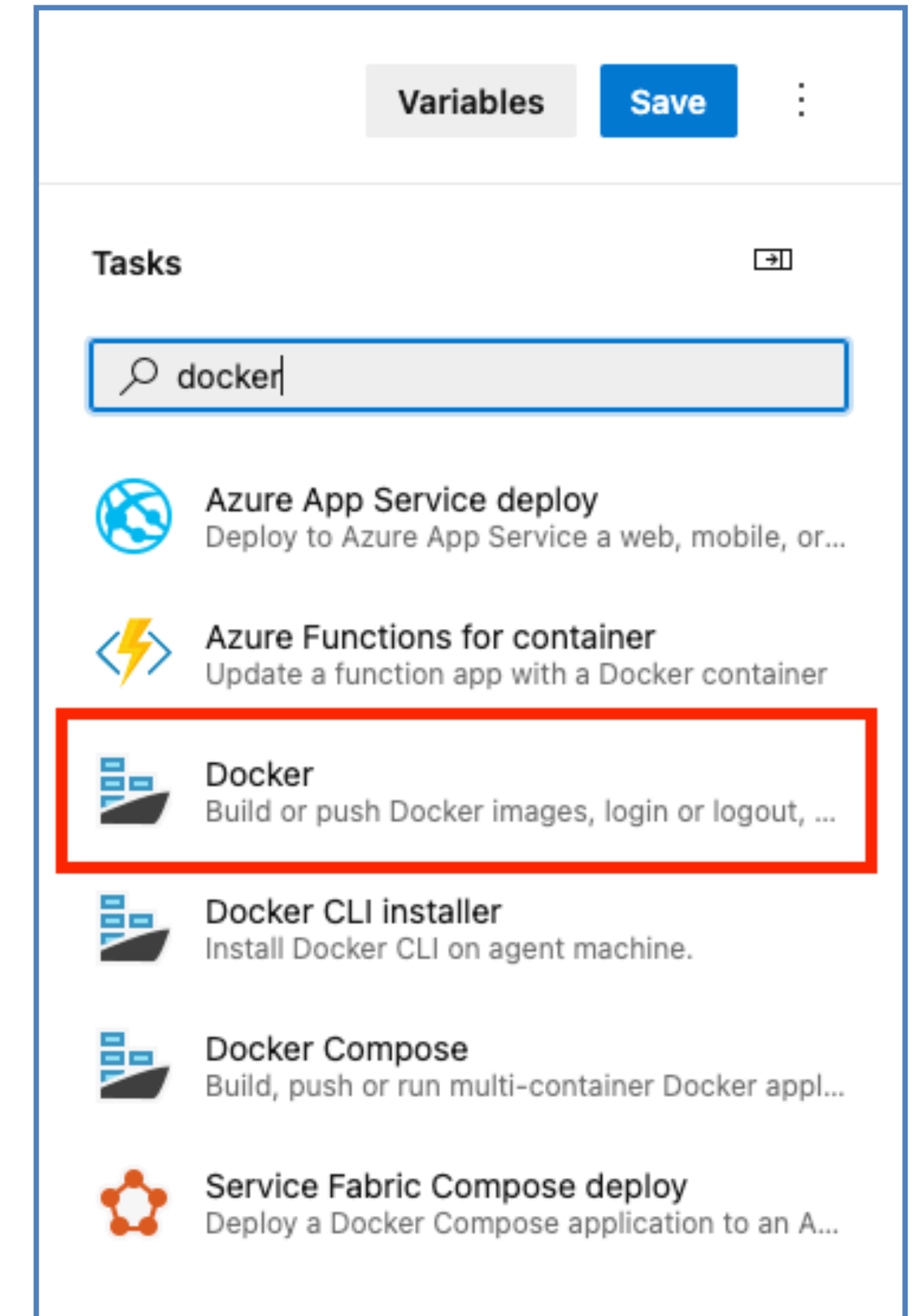
- Pull up the wizard, which has all plugins.





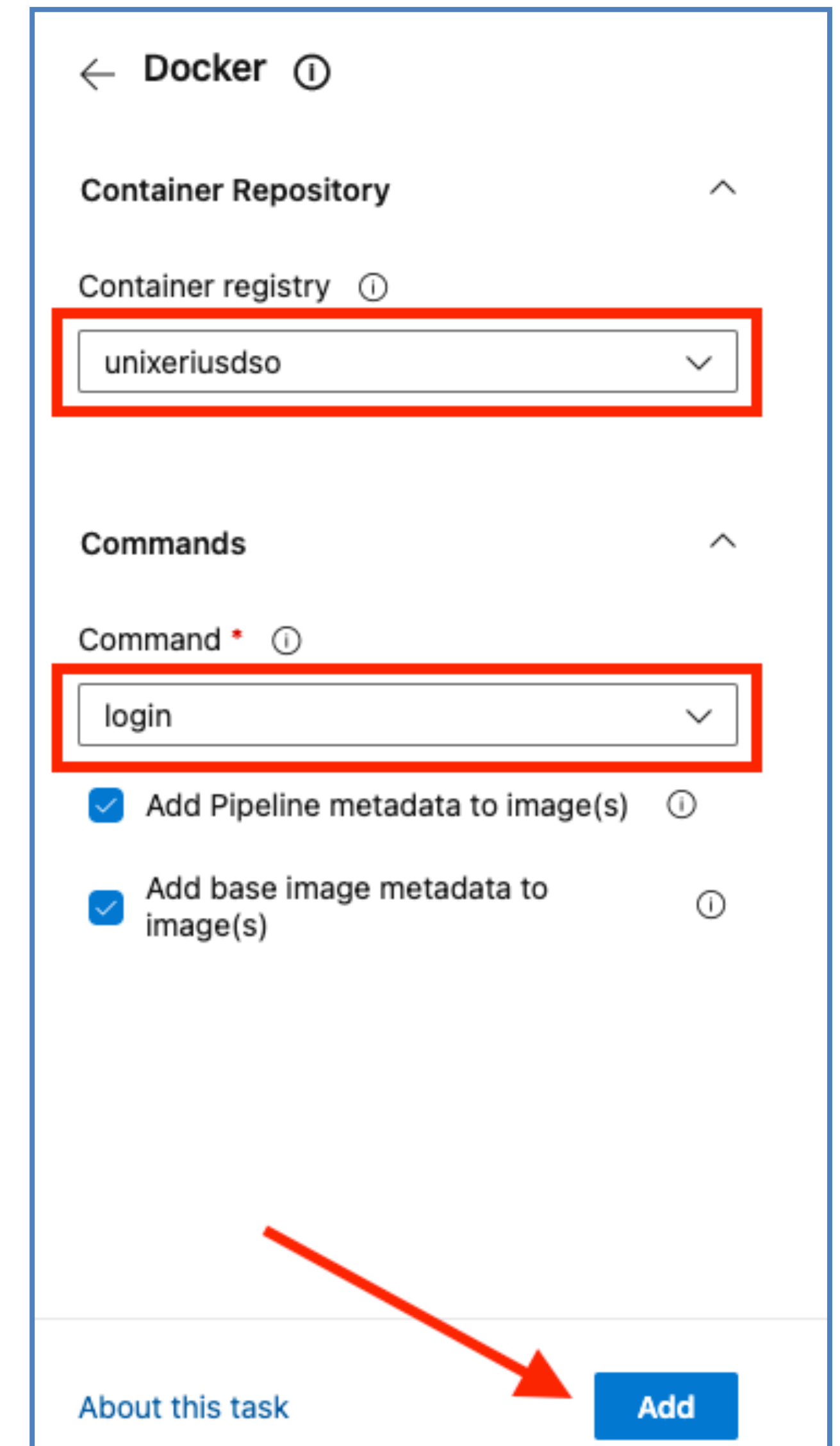
# Docker login / logout

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



# Docker login / logout

- Chose "*Login*" as command.
- Chose the "*unixeriusdso*" registry.
- Repeat for "*Logout*".



The screenshot shows the Docker configuration interface for a task. It includes a back arrow and the title "Docker" with an information icon. Under the "Container Repository" section, the "Container registry" dropdown is set to "unixeriusdso" and is highlighted with a red box. Under the "Commands" section, the "Command" dropdown is set to "login" and is also highlighted with a red box. Below these, there are two checked options: "Add Pipeline metadata to image(s)" and "Add base image metadata to image(s)". At the bottom, there is a link "About this task" and a blue "Add" button, which is pointed to by a red arrow.

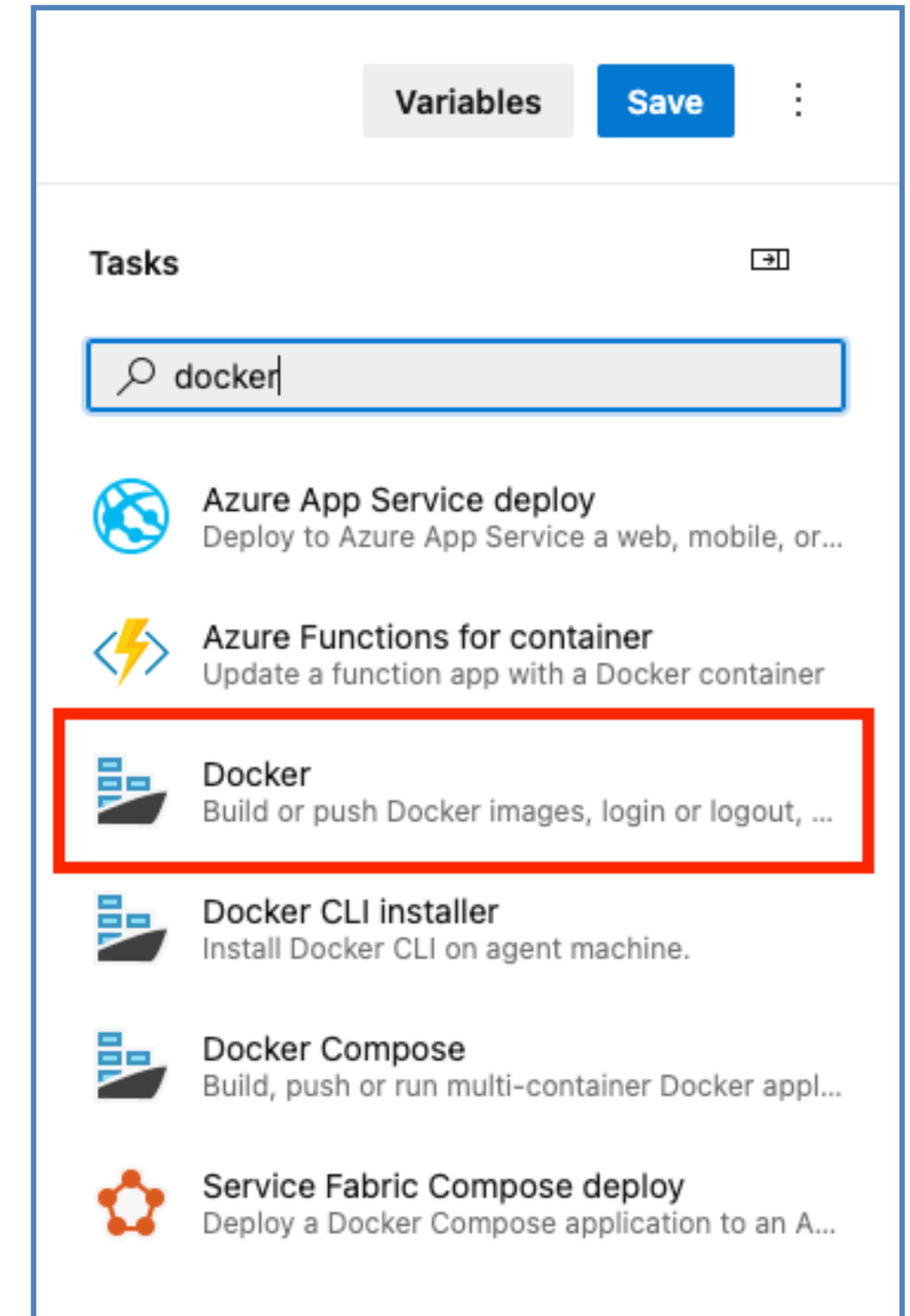
# Docker login / logout

- Your code should be ->
- Save and run, to test.

```
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://unixeriUSDso-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.



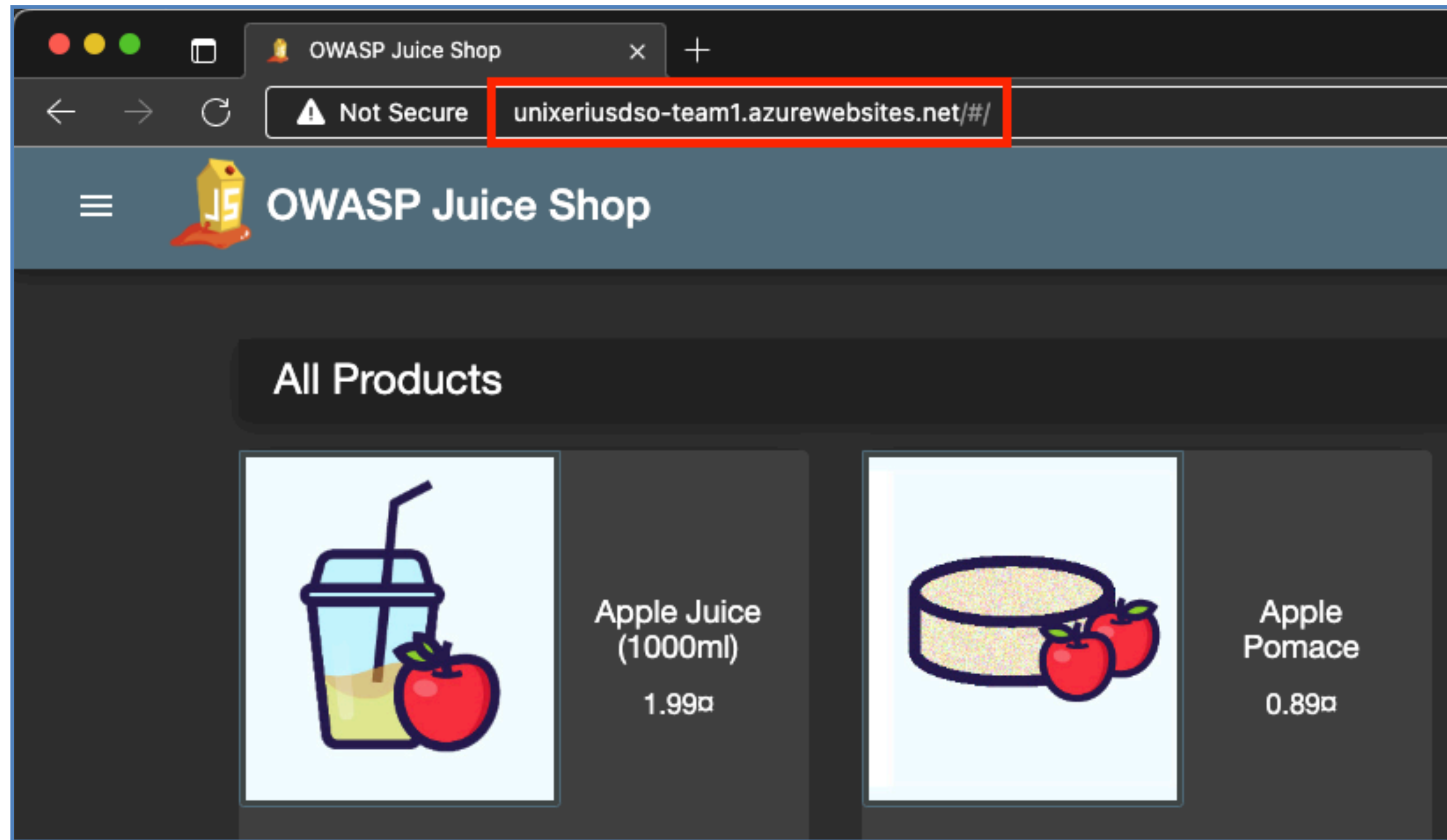
# The code

- 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?



# Our "bingo" card

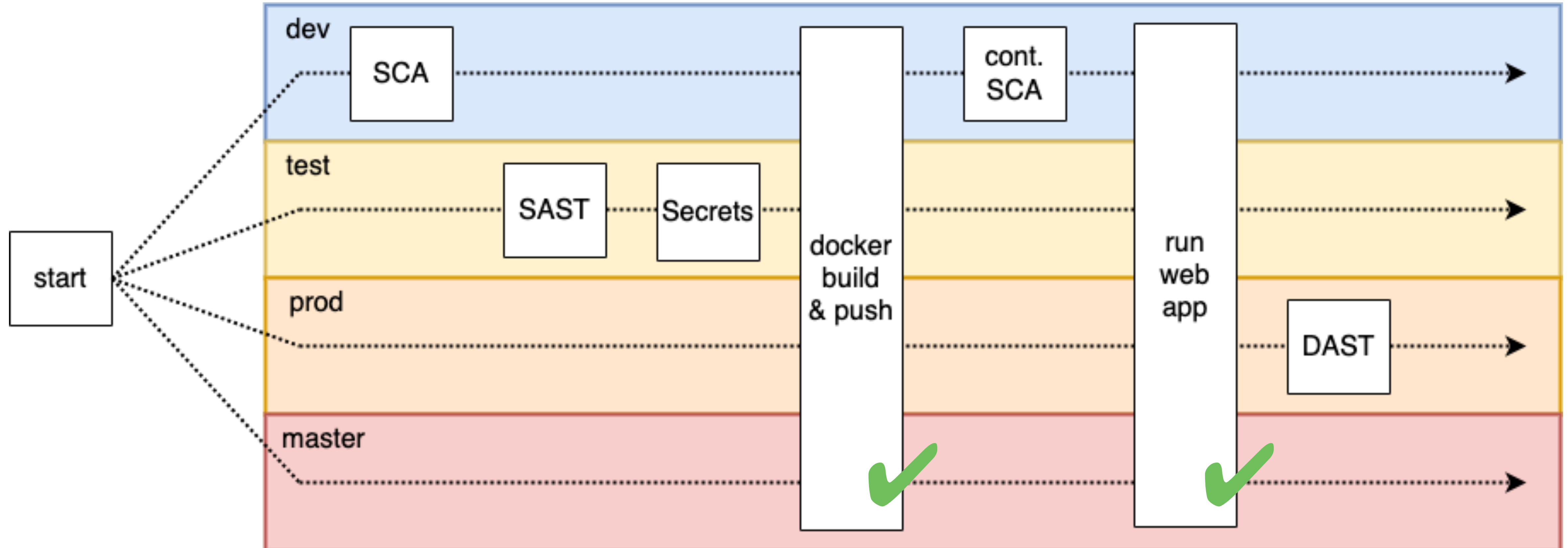
DevOps	Intro	Work IRL	AzDO	SDLC	Juice!	
DevOps	Agile	Git	Virtual	Contain	CI/CD	
DevSecOps	App test	DSO	Vulns	SCA	TModel	
DevSecOps	SAST	Secrets	PROD			
DevSecOps	VulnScan	Pentest	DAST	WAF		

# 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



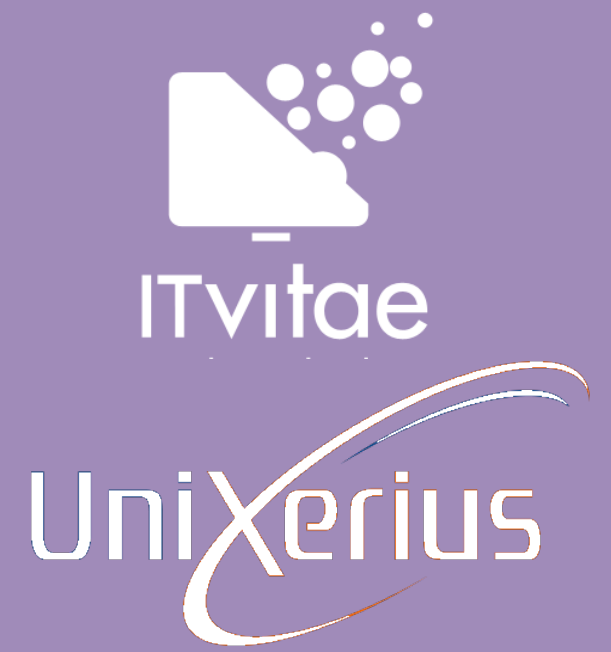
# What have we achieved?

- 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.

# 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](#)