

The background features a vibrant, abstract design. On the left, there are horizontal, wavy bands of color in shades of red, orange, yellow, and green. On the right, a bright white light source emits a series of colorful rays in shades of blue, cyan, and yellow, creating a sunburst effect.

CISCO *Live!*

Let's go

#CiscoLiveAPJC



The bridge to possible

Introduction to Continuous Development (CI/CD) with GitLab

Hank Preston, Principal Engineer Learning and Certifications
ccie 38336 | devnet expert 20220001
@hfpreston
DEVNET-1723

cisco *Live!*

#CiscoLiveAPJC

Cisco Webex App

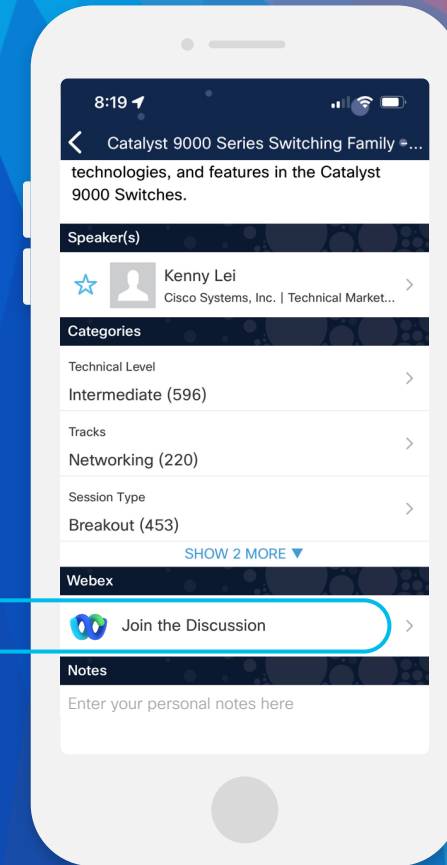
Questions?

Use Cisco Webex App to chat with the speaker after the session

How

- 1 Find this session in the Cisco Live Mobile App
- 2 Click “Join the Discussion”
- 3 Install the Webex App or go directly to the Webex space
- 4 Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until December 22, 2023.



<https://ciscolive.ciscoevents.com/ciscolivebot/#DEVNET-1723>



Checkout the demo app
<http://3.142.250.188/>

Username:

- customer, operator, or sysadmin

Password: password



Checkout the pipeline
<http://bit.ly/hankdemo-cicd>

Submit Request Manage Services Network Request Application Hello sysadmin Log Out

Network Service Management

Here you can view and manage all network service requests that have been made.

VLAN Name	VLAN Id	Description	Submitter	Status	Action
Dev01	701	Dev Service 01	devdata	submitted	<button>Approve</button> <button>Deny</button>
Dev02	702	Dev Service 02	devdata	submitted	<button>Approve</button> <button>Deny</button>
Dev03	703	Dev Service 03	devdata	approved	<button>Approve</button> <button>Deny</button>
Dev04	704	Dev Service 04	devdata	approved	<button>Approve</button> <button>Deny</button>
Dev05	705	Dev Service 05	devdata	denied	<button>Approve</button> <button>Deny</button>
ReadmeDemo_1	None	A new network for the readme doc.	customer	submitted	<button>Approve</button> <button>Deny</button>

Merge branch 'staging' into 'main'

Update main with CID changes

See merge request 12

10 jobs for main (queued for 1 second)

latest

15ac5c86

No related merge requests found.

Pipeline

Needs Jobs 10 Tests

Group jobs by Stage Job dependencies

Validate	Test	Build-app	Build-host	Deploy-host
☒ format-check	☒ run_pytest	☒ build_image	☒ build-production-host	☒ deploy-production-host
☒ lint				
☒ tf-format				
☒ tf-validate				

Agenda

- What exactly are we trying to “CICD”?
- Demo upfront, business in the back
- How it’s made – dissecting the pipeline
- Ending with a dash of reality about CICD

What's out of scope for the session

- GitLab server / project setup
- GitLab runner setup
- Specifics about
 - Linting or validating code
 - Writing and running tests
 - Creating Docker images
 - Deploying hosts to the cloud
 - Running containers on hosts

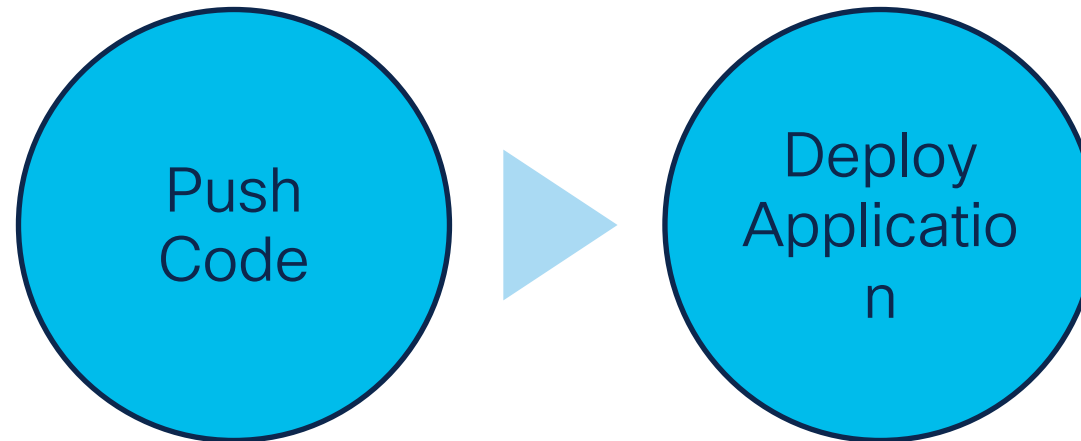
All code available

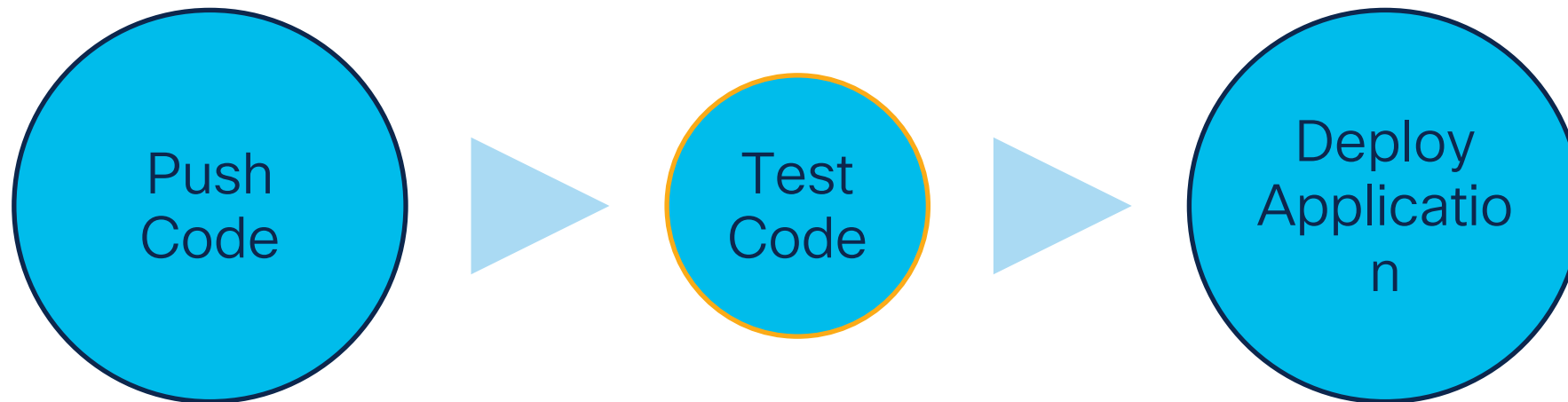
- [Project Repository](#)
- A functional demo application created for this session
 - Written in Python
 - Test cases with pytest
 - Containerized in Docker
 - Web application
 - Deployed to cloud with Terraform
- A functional CI/CD Pipeline built and operational
 - Pipeline logs, history, environments, etc available
 - [Detailed video of working pipeline](#)

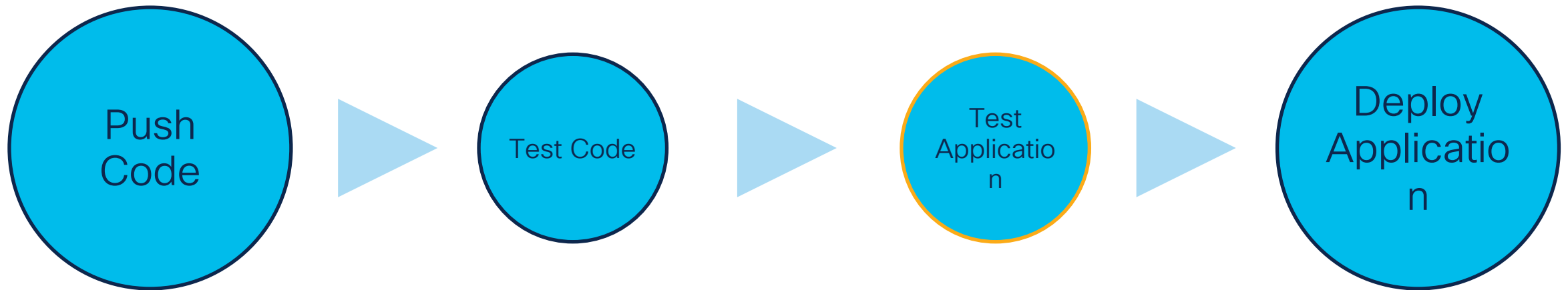
See official docs

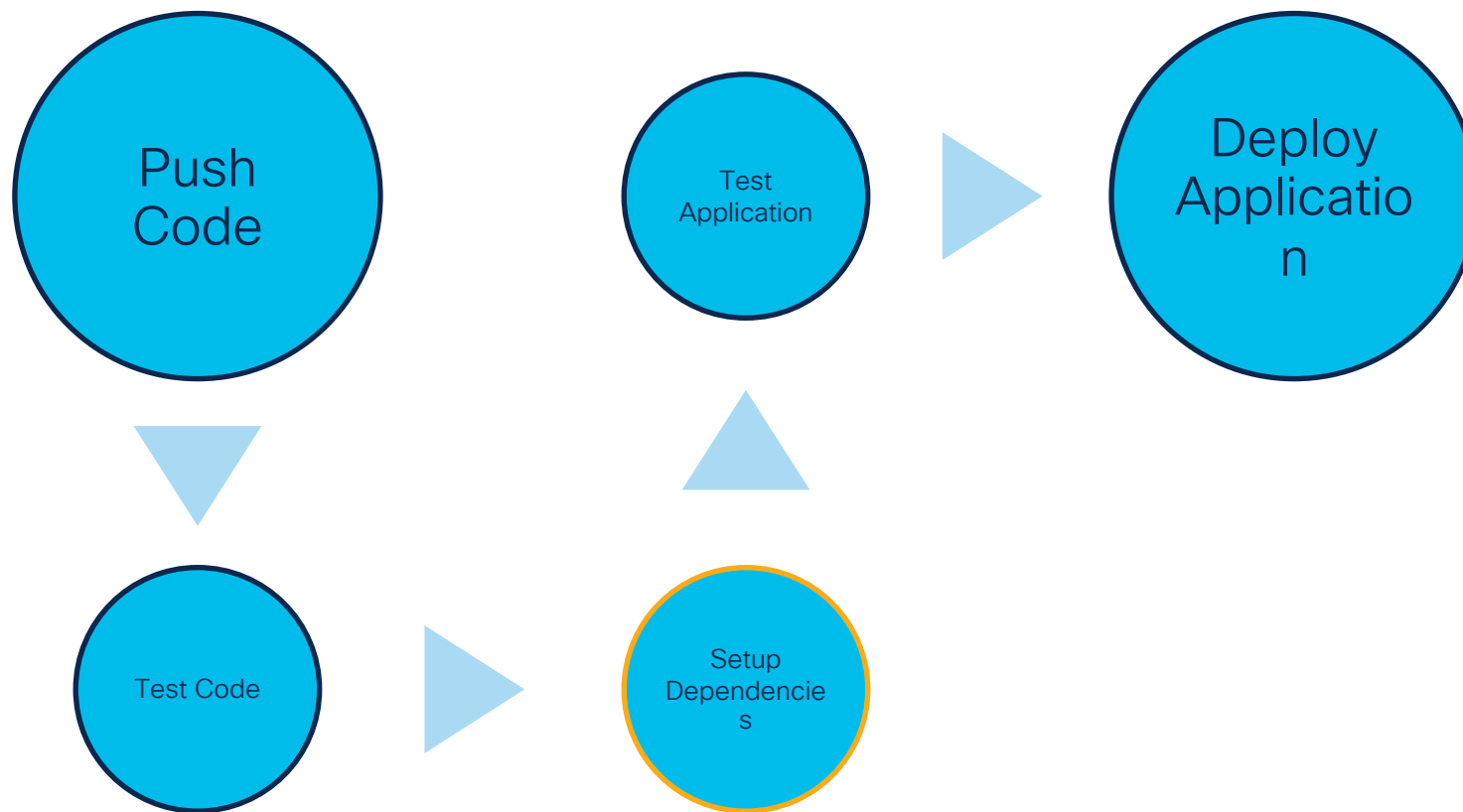
- To learn more or dive deeper, see [GitLab CI/CD Docs](#)
 - They are really quite good
- Effort made to provide links within presentation to relevant content
- Any discrepancy between my slides and documentation indicates a fault of mine. Docs always win

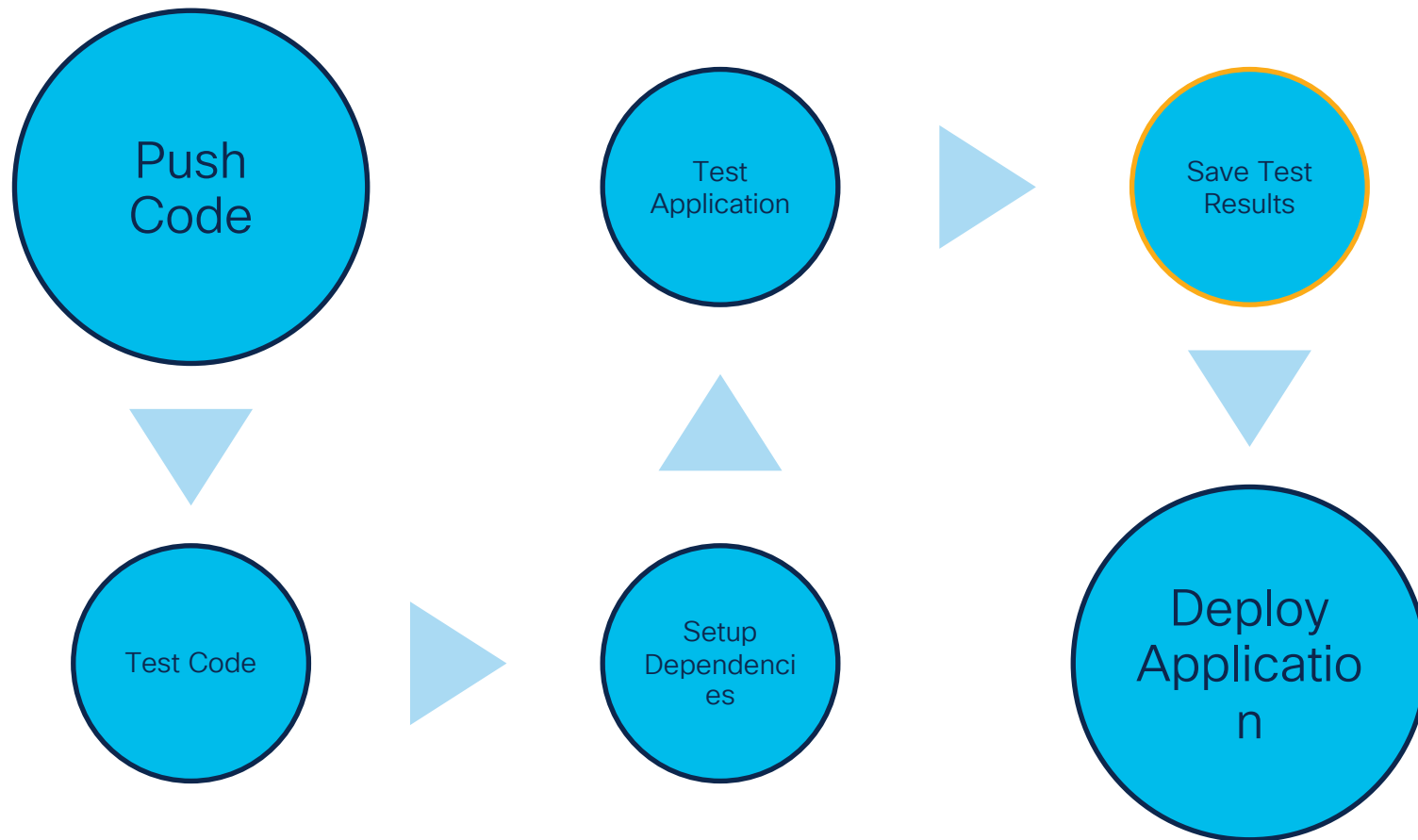
What exactly are we trying
to “CICD”?

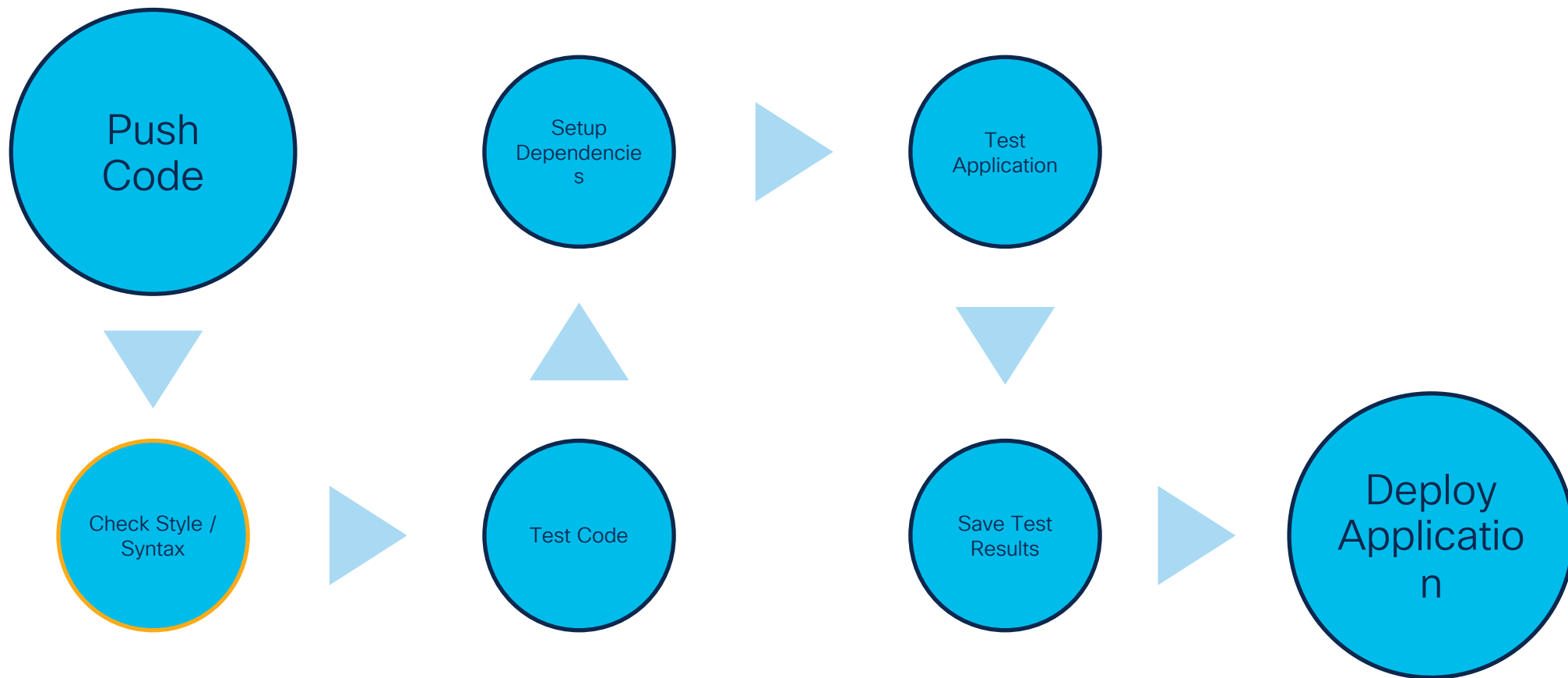


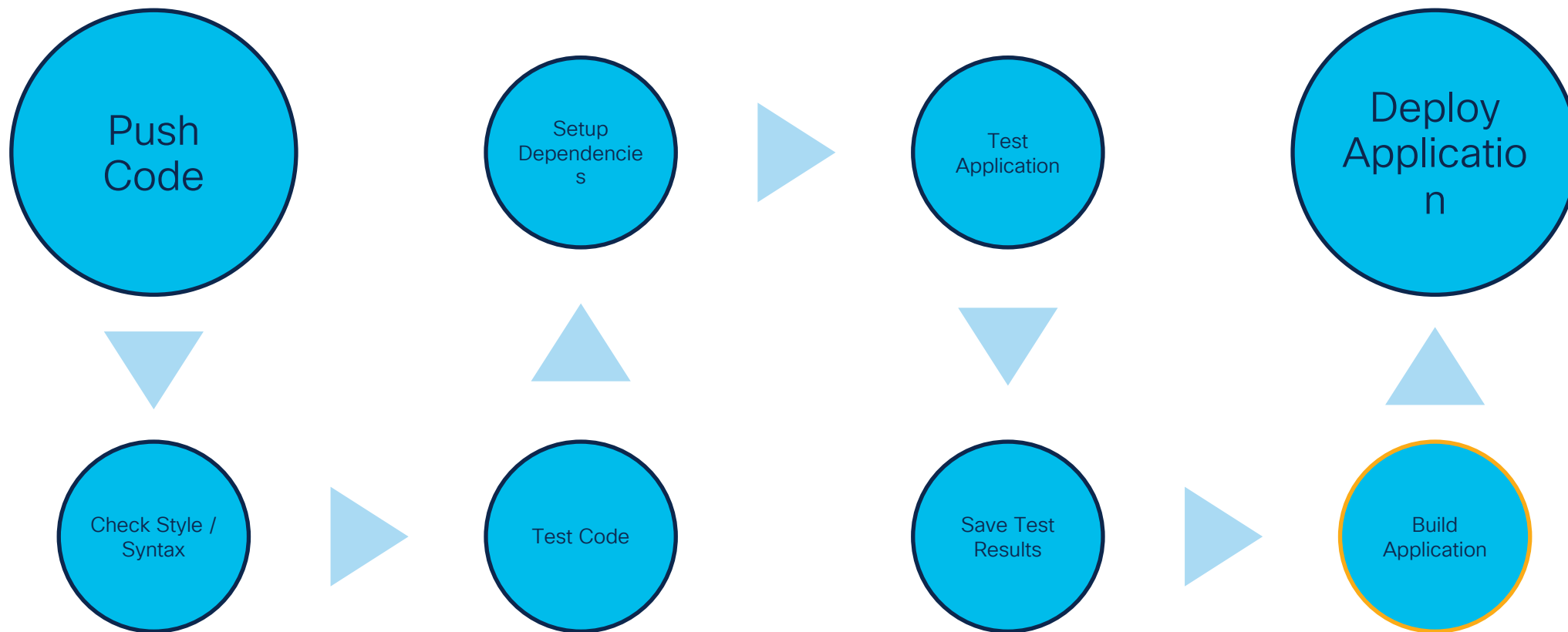


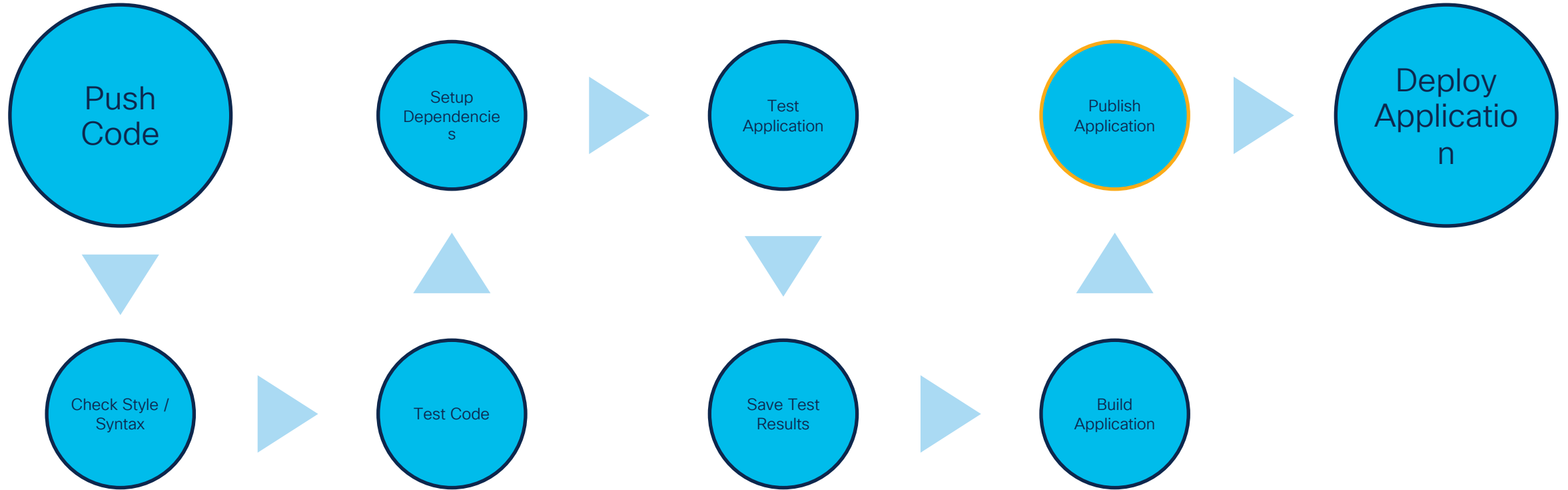


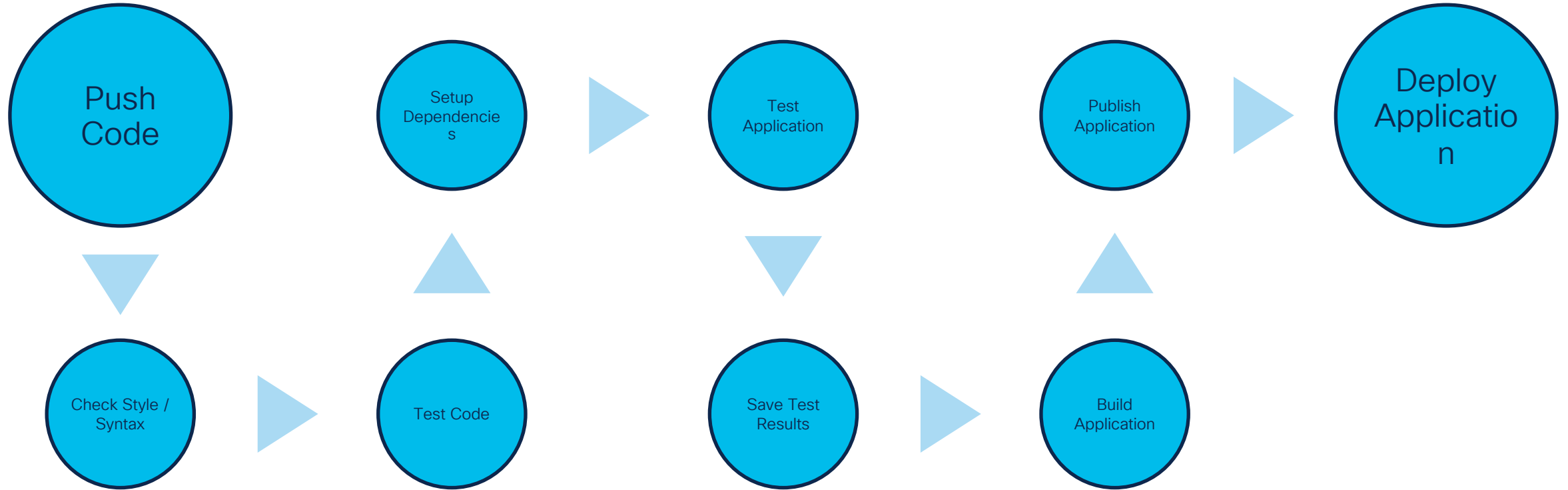












Wait... we need to manage Dev > Test > Prod differently...

Demo upfront,
business in the
back

GitLab Pipeline Updating Production

Merge branch 'staging' into 'main'

Update main with CICD changes

See merge request !2

🕒 10 jobs for main (queued for 1 second)

🚩 latest

🔗 15ac5c86

🔗 No related merge requests found.

Pipeline

Needs

Jobs10

Tests0

Group jobs by

Stage

Job dependencies

Validate

format-check

lint

tf-format

tf-validate

Test

run_pytest

Build-app

build_image

Build-host

build-production-host

Deploy-host

deploy-production-host

Build-docker

build-production-docker

Deploy-docker

deploy-production-docker

Demo Video!!!

Full (longer) demo video of
pipeline available as well

How it's made – dissecting the pipeline

.gitlab-ci.yml

- The pipeline instruction file
 - Located at root of the project
- Written in YAML with [GitLab syntax](#)
- Can "[include](#)" other files for modularity

Pipeline Stages

- “[stages](#)” define parts of the overall process
- Default Stages
 - build, test, deploy
- Fully customizable stage
 - How many stages
 - Names of stages
- All jobs in a stage will run at the same time*
- All jobs in a stage must complete before the next stage’s jobs start

* Unless there is another requirement delaying one job

Defining Stages for My Pipeline

- Prep
 - Things that need to be done before or in support of pipeline
- Validate
 - Verify code and files. Linting, format checks, etc.
- Test
 - Run all code tests. Unit and Integration
- Build-App
 - Create all application artifacts. Containers, binaries, etc.

stages:

- prep
- validate
- test
- build-app
- build-host
- deploy-host
- build-docker
- deploy-docker
- cleanup

Defining Stages for My Pipeline

- Build-Host
 - Create Terraform plan to deploy host to run application
- Deploy-Host
 - Apply Terraform plan to create the host
- Build-Docker
 - Create Terraform plan to start Docker containers
- Deploy-Docker
 - Apply Terraform plan to start application
- Cleanup
 - Terraform destroy to teardown application

stages:

- prep
- validate
- test
- build-app
- build-host
- deploy-host
- build-docker
- deploy-docker
- cleanup

*Hold on a minute...
Where are the other stages?*

Pipeline Needs Jobs 10 Failed Jobs 2 Tests 12

Group jobs by Stage Job dependencies

Validate	Test	Build-app	Cleanup
<div>✓ format-check ↺</div>	<div>✓ run_pytest ↺</div>	<div>✓ build_image ↺</div>	<div>✗ destroy-production-docker ↺</div>
<div>✓ lint ↺</div>			<div>✓ destroy-production-host ↺</div>
<div>✓ tf-format ↺</div>			<div>✗ destroy-staging-docker ↺</div>
<div>✓ tf-validate ↺</div>			<div>✓ destroy-staging-host ↺</div>

Pipeline Job

- [Jobs](#) are steps in the pipeline
- Occur during one stage
- Run in a dedicated Docker container
- Contain rules determining when a job runs
- Must have a “[script](#)” block defining what happens during the job
- Can succeed, fail, or be skipped*
- Can create artifacts saved or used in other jobs

* Other job statuses include pending, running, canceled, manual, etc

Job Definition: Build and Push a Docker Image

- Each job runs in a dedicated container using the “[image](#)”
- “[services](#)” identify additional containers required during job*
- “[variables](#)” can be defined in many places
 - Pipeline, Job, Conditionally, Project Level
 - Many [predefined variables](#) exist containing details about the job/project
- “[script](#)” is the list of commands to run for job
 - Commands must succeed or job fails
- “[rules](#)” determine when to run a job

```
tac_plus_image:
  stage: prep
  image: docker:stable
  services:
    - docker:dind
  variables:
    IMAGE_TAG: $CI_REGISTRY_IMAGE/tac_plus:$CI_COMMIT_REF_SLUG
  script:
    - cd tac_plus
    - |
      docker login -u $CI_REGISTRY_USER \
                    -p $CI_REGISTRY_PASSWORD \
                    $CI_REGISTRY
    - docker info
    - docker build -t $IMAGE_TAG .
    - docker push $IMAGE_TAG
  # Only run if there have been changes in the tac_plus build
  rules:
    - changes:
      - tac_plus/*
```

* docker:dind (docker in docker) allows the docker client in a container to interact with the host server's docker service

Variables

- “[variables](#)” provide dynamic input into pipeline
 - Credentials/Secrets (strings or files)
 - Project details (name, branch, commit)
- Defined in
 - Pipeline, Job, part of condition
 - Predefined variables from GitLab
 - At Project, Group, or Instance Level*
- Used in many parts of pipeline
 - `docker login -u $CI_REGISTRY_USER`
 - `if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH \&\& $DEPLOY_APP == "True"`

* Very useful for secrets

GitLab Variable Creation

- Excellent place for secrets
- “Protected” variables only available on “protected” branches
 - Prevent merge requests from unknown developers accessing secrets
- “Masked” will hide value of variable in pipeline and job logs

Type	↑ Key	Value	Protected	Masked	Environments	
Variable	AWS_ACCESS_KEY_ID 	***** 	✓	✓	All (default)	
Variable	AWS_REGION 	***** 	✓	✓	All (default)	
Variable	AWS_SECRET_ACCESS... 	***** 	✓	✓	All (default)	
Variable	DEPLOY_APP 	***** 	✓	✗	All (default)	
File	TF_VAR_PROD_SSH_KEY 	***** 	✓	✗	All (default)	

[Project > Settings > CI/CD > Variables](#)

Rules determine when a job runs

- Not all jobs should run on every pipeline execution
- Common conditions
 - What branch is the pipeline running on?
 - Why is the pipeline running? Push, Merge Request, Schedule?
 - Did a relevant file change?
 - Value of a variable?
- Can set values of variables based on a rule

Example rules definitions

- Sometimes a job should only run [if files have changed](#)
 - Also “[exists](#)” rules to check for a file
- Rules based on a variable value is quite common
 - Can use all types of variables
 - “[if](#)” clauses support complex logic (&& / ||)
- Multiple “rules” on a job are treated as “or”
- Set variables based on a rules for more dynamic jobs

```
tac_plus_image:
  stage: prep
  # Only run if there have been changes in the tac_plus build
  rules:
    changes:
      - tac_plus/*

build_image:
  stage: build-app
  rules:
    # Only build and push if on specific branches of the project
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
      variables:
        environment: production
    - if: $CI_COMMIT_BRANCH == "staging"
      variables:
        environment: staging

# An example rule based on source of pipeline run
example_schedule:
  script: echo "This job only runs on a schedule"
  # Only run if a scheduled job
  rules:
    - if: $CI_PIPELINE_SOURCE == "schedule"
      # Other options: web, merge_request, pushes
```

“when” to run a job

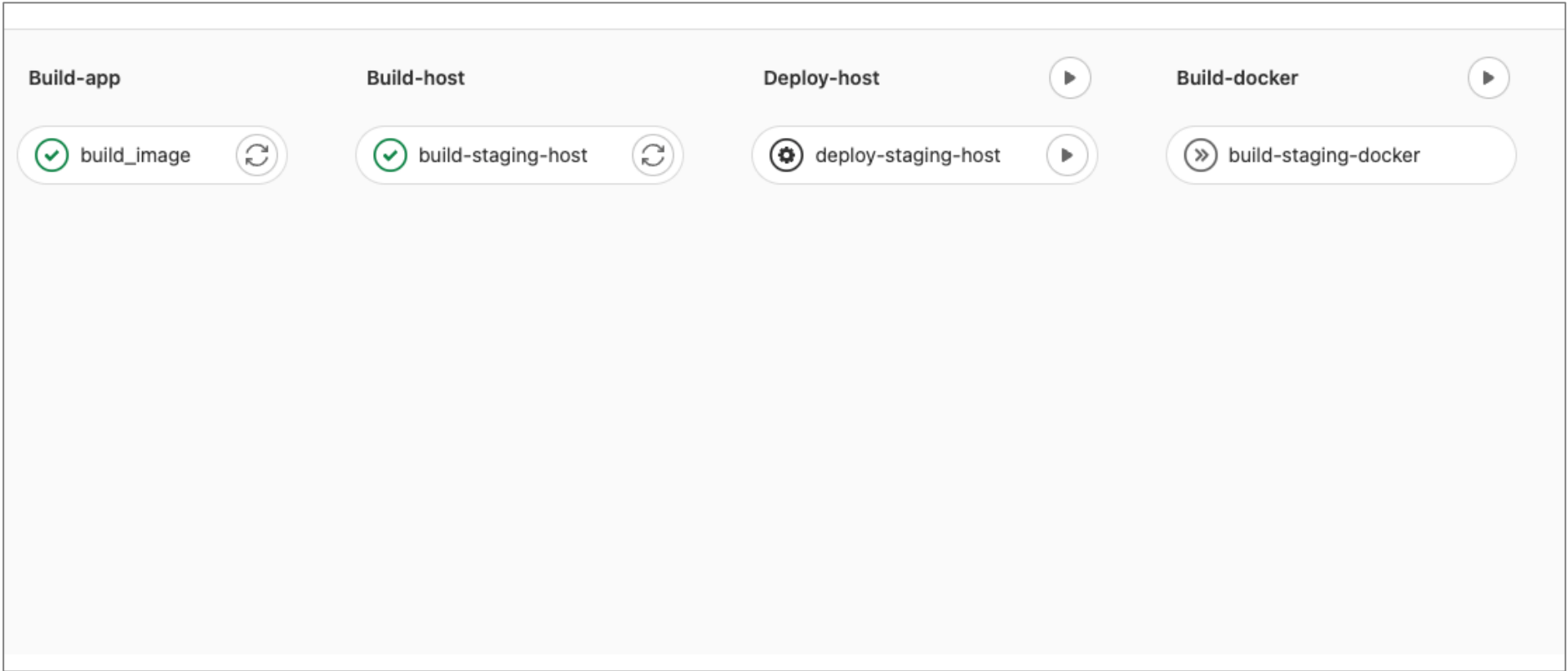
- “on_success” (default)
 - Run when all jobs in earlier stages succeed*
- “on_failure”
 - Run when at least one job in earlier stage fails
- “always”
 - Run.. Always
- “manual”
 - Require a manual trigger to execute
- “delayed”
 - Wait for some period of time before running job
 - “start_in” a time period (ex. 5 seconds, 1 day)
- “never”
 - Only used in a rule to create a prevention rule

Pausing a Pipeline for Manual Intervention

- While automation is great, sometimes you need (want) a person to trigger a part of pipeline
- Deployments to production are common reasons for this

```
.tf-deploy-host:
  image: terraform-images/releases/terraform:1.1.9
  stage: deploy-host
  variables:
    TF_VAR_ENVIRONMENT: ${ENVIRONMENT_NAME}
    TF_STATE_NAME: ${ENVIRONMENT_NAME}
  environment:
    name: ${ENVIRONMENT_NAME}
  resource_group: ${ENVIRONMENT_NAME}
  script:
    - cd "${TF_ROOT}"
    - source gitlab_tf_env
    - terraform init -input=false -reconfigure
    - |
      terraform apply -input=false \
        -target="module.ec2_instance" \
        -target="module.security_group" \
        -target="module.vpc" \
        "${TF_PLAN_CACHE}"
  when: manual
```

Note: the image used in this example is abbreviated to fit in code example.
Full image is: "registry.gitlab.com/gitlab-org/terraform-images/releases/terraform:1.1.9"



Artifacts and Cache

- Capture and store content from a job
- Cache
 - Commonly used for dependencies such as packages downloaded
 - Stored on the runner
 - Jobs must run on same runner
- Artifacts
 - Commonly used for things created by job
 - Stored in GitLab
 - Can be downloaded from GitLab from anywhere
 - Will expire
 - Useful for test reports/results
 - Define job “dependencies” to use artifacts in another job

Saving test results with Artifacts

- Standard output/error saved in job log
- But detailed log files or reports must be saved as artifacts or be lost
- “[paths](#)” will save a specific file(s) to GitLab for download
- “[reports](#)” render reports within GitLab Job

```
run_pytest:
  stage: test
  image: python:3.9
  services:
    - name: $CI_REGISTRY_IMAGE/tac_plus:main
      alias: tac_plus
  variables:
    TACACS_HOST: tac_plus
    TACACS_SECRET: labtacacskey
  script:
    - pip install -e .
    - pip install -e '[test]'
    - pytest --junitxml=report.xml
  # Save test logs for later detailed review
  # - The raw pytest.log is saved for download
  # - The unit test report is displayed in GitLab
  artifacts:
    when: always
    name: "$(date +%Y%m%d_%H%M%S)-pytest.log"
    paths:
      - pytest.log
    reports:
      junit: report.xml
```

128 WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: <https://pip.pypa.io/warnings/venv>

129 WARNING: You are using pip version 22.0.4; however, version 22.1.1 is available.

130 You should consider upgrading via the '/usr/local/bin/python -m pip install --upgrade pip' command.

131 \$ pytest --junitxml=report.xml

132 ===== test session starts =====

133 platform linux -- Python 3.9.13, pytest-7.1.2, pluggy-1.0.0

134 rootdir: /builds/hpreston/demo-python-tacacs, configfile: pytest.ini

135 collected 12 items

136 tests/test_auth.py [33%]

137 tests/test_home.py . [41%]

138 tests/test_services.py ... [66%]

139 tests/test_tacacsc.py [100%]

140 ----- generated xml file: /builds/hpreston/demo-python-tacacs/report.xml -----

141 ===== 12 passed in 0.21s =====

143 Uploading artifacts for successful job 00:04

144 Uploading artifacts...

145 pytest.log: found 1 matching files and directories

146 Uploading artifacts as "archive" to coordinator... 201 Created id=2516761625 responseStatus=201 Created token=fu3xB183

147 Uploading artifacts...

148 report.xml: found 1 matching files and directories

149 Uploading artifacts as "junit" to coordinator... 201 Created id=2516761625 responseStatus=201 Created token=fu3xB183

151 Cleaning up project directory and file based variables 00:01

153 Job succeeded

run_pytest

Duration: 1 minute 2 seconds
Finished: 4 days ago
Timeout: 1h (from project)
Runner: #12270837 (J2nyww-s) 4-blue.shared.runners-manager.gitlab.com/default
Tags: docker

Job artifacts
These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.
Keep Download Browse

Commit 15ac5c86
Merge branch 'staging' into 'main'

Pipeline #550054556 for main
test

run_pytest

Pipeline Needs Jobs 10 Tests 12

< run_pytest

12 tests 0 failures 0 errors 100% success rate 137.00ms

Tests

Suite	Name	Filename	Status	Duration	Details
tests.test_auth	test_login_user		✓	31.00ms	View details
tests.test_auth	test_invalid_login		✓	25.00ms	View details
tests.test_auth	test_logout		✓	24.00ms	View details
tests.test_auth	test_get_login		✓	22.00ms	View details
tests.test_home	test_get_homepage		✓	20.00ms	View details
tests.test_tacacsc	test_authenticate_01		✓	3.00ms	View details
tests.test_tacacsc	test_authenticate_02		✓	3.00ms	View details

Scripting the action

- “[script](#)” defines what command(s) are executed for the job
- Mandatory for all jobs*
- [Exit codes](#) other than “0” result in job failure
- The output from the script available in job log
 - Features like echo, [color codes](#), and [sections](#) can organize log

* Except for “trigger jobs” which are out of scope for this introduction

Before and After Scripts

- [“before_script”](#)
 - Commands that are run before the script block of a job
 - Commands run in same shell as “script”
 - Any artifacts loaded from previous jobs are loaded first
 - Can affect job success/failure
- [“after_script”](#)
 - Commands run after the script block
 - Run even for failed jobs
 - Run in dedicated shell from before_script and script commands
 - Doesn't affect job success/failure

Job Script Example

- “[before_script](#)” used for preparation steps
 - For example, logging into Docker registry
- “[script](#)” runs the commands needed for job
 - Hint... whatever you would run manually to do the work is what goes in the script

```
build_image:
  stage: build-app
  image: docker:stable
  services:
    - docker:dind
  variables:
    IMAGE_TAG: |
      $CI_REGISTRY_IMAGE/network-request:$CI_COMMIT_REF_SLUG
  before_script:
    - |
      docker login -u $CI_REGISTRY_USER \
        -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker info
  script:
    - docker build -t $IMAGE_TAG .
    - docker push $IMAGE_TAG
  rules:
    # Only build and push if on "main" or "staging"
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
    - if: $CI_COMMIT_BRANCH == "staging"
```

1 Running with gitlab-runner 14.10.0~beta.50.g1f2fe53e (1f2fe53e)

2 on blue-4.shared.runners-manager.gitlab.com/default J2nyww-s

3 **Preparing the "docker+machine" executor** 00:24

4 Using Docker executor with image docker:stable ...

5 Starting service docker:dind ...

6 Pulling docker image docker:dind ...

7 Using docker image sha256:1f6c0346b90562fa44a32011666df7a7f413ee260a0510b52ca974bef4d50f15 for docker:dind with digest docker@sha256:4e6110013c1d73f65f0635a2c498c3e59105efc092c1a930bbc480391b6e0f24 ...

8 Waiting for services to be up and running...

9 Pulling docker image docker:stable ...

10 Using docker image sha256:b0757c55a1fdbb59c378fd34dde3e12bd25f68094dd69546cf5ca00ddbba7a33 for docker:stable with digest docker@sha256:fd4d028713fd05a1fb896412805daed82c4a0cc84331d8dad00cb596d7ce3e3a ...

12 **Preparing environment** 00:02

13 Running on runner-j2nyww-s-project-34435628-concurrent-0 via runner-j2nyww-s-shared-1654046128-6dea0fba...

15 **Getting source from Git repository** 00:01

16 \$ eval "\$CI_PRE_CLONE_SCRIPT"

17 Fetching changes with git depth set to 20...

18 Initialized empty Git repository in /builds/hpreston/demo-python-tacacs/.git/

19 Created fresh repository.

20 Checking out 87892417 as staging...

21 Skipping Git submodules setup

23 **Downloading artifacts** 00:01

24 Downloading artifacts for run_pytest (2530543393)...

25 Downloading artifacts from coordinator... ok id=2530543393 responseStatus=200 OK token=cPE_UsNU

27 **Executing "step_script" stage of the job script** 00:44

28 Using docker image sha256:b0757c55a1fdbb59c378fd34dde3e12bd25f68094dd69546cf5ca00ddbba7a33 for docker:stable with digest docker@sha256:fd4d028713fd05a1fb896412805daed82c4a0cc84331d8dad00cb596d7ce3e3a ...

29 \$ chmod 0600 \${TF_VAR_PROD_SSH_KEY}

30 \$ docker login -u \$CI_REGISTRY_USER -p \$CI_REGISTRY_PASSWORD \$CI_REGISTRY

31 WARNING! Using --password via the CLI is insecure. Use --password-stdin.

32 WARNING! Your password will be stored unencrypted in /root/.docker/config.json.

33 Configure a credential helper to remove this warning. See

34 <https://docs.docker.com/engine/reference/commandline/login/#credentials-store>

35 Login Succeeded

36 \$ docker info

37 Client:

38 Debug Mode: false

39 Server:

40 Containers: 0

41 Running: 0

42 Paused: 0

43 Stopped: 0

build_image

Duration: 1 minute 15 seconds

Finished: 11 minutes ago

Timeout: 1h (from project)

Runner: #12270837 (J2nyww-s) 4-blue.shared.runners-manager.gitlab.com/default

Tags: docker

Commit 87892417
fix needs

Pipeline #552704551 for staging

build-app

→ build_image

git init

before_script

script

```
83 Live Restore Enabled: false
84 Product License: Community Engine
85 $ docker build -t $IMAGE_TAG .
86 Step 1/11 : FROM python:3.9
87 3.9: Pulling from library/python
88 e756f3fdd6a3: Pulling fs layer
89 bf168a674899: Pulling fs layer
90 e604223835cc: Pulling fs layer
91 6d5c91c4cd86: Pulling fs layer
92 2cc8d8854262: Pulling fs layer
93 2767dbfeeb87: Pulling fs layer
94 dec404173396: Pulling fs layer
95 20cadcl653df: Pulling fs layer
96 fa2afb411b49: Pulling fs layer
97 6d5c91c4cd86: Waiting
98 2cc8d8854262: Waiting
```

Tips for better pipelines

Use defaults to simplify

- Pipeline “[defaults](#)” set values for jobs that can be overridden if needed*
- Common examples
 - “image” great when most jobs use same base image
 - “tags” to control which runners a job can use
 - “before_” and “after_” scripts
 - “services” for supporting containers
 - “artifacts” and “cache”

* Not all job settings can be set as defaults

Match jobs to runners with tags

- All runners are not necessarily the same
- Jobs might require different features
 - Linux or Windows
 - Docker
 - Special libraries or tools
 - Access to private data shares
- “[tags](#)” used to identify [features of a runner and requirements of a job](#)
 - All tags on a job must be available on a runner
 - A runner can have “extra tags” and still be valid

Pipeline defaults

- Ensure jobs are assigned to “docker” enabled runners
- Prepare a “[File](#)” Secret for use as SSH key

```
default:
  # Tag all jobs as "docker" to ensure they are placed onto
  # valid GitLab Runners
  tags:
    - docker
  before_script:
    # The SSH key for logging into the host server is a
    # Protected/Masked File Based
    # GitLab CI/CD Variable. Need to set the permissions
    # correctly to allow use of the key to log into
    # and manage the deployed host server.
    - chmod 0600 ${TF_VAR_PROD_SSH_KEY}
```

Include other pipeline files

- Add jobs, defaults, variables, etc from other files into your pipeline with “[include](#)”
- Reasons
 - Improve readability
 - Reduce duplication
- Sources for inclusion
 - “[local](#)” – from same project
 - “[file](#)” – from another project on GitLab server
 - “[remote](#)” – from a HTTP(s) GET request
 - “[template](#)” – a [GitLab CI/CD template](#)
- Included files are merged and processed

"include" example

- For readability I broke my pipeline into multiple files
- "rules" can be used to determine when to include a file

```
include:
- local: .gitlab-ci.tacplus.yml
  rules:
    # Only build new images of tacplus on specific
    # branches of project
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
    - if: $CI_COMMIT_BRANCH == "staging"

- local: .gitlab-ci.terraform.yml
  rules:
    # Deployments are only done on specific branches of
    # the project
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
    - if: $CI_COMMIT_BRANCH == "staging"
```

Managing Deployments (briefly)

Note: Deployments and Environments could/should be a dedicated session.

Deployments and Environments

- Environments are a place where code is deployed
 - Production, Staging, Test, etc
- A “[deploy job](#)” is any job that includes a targeted environment
- A “[resource_group](#)” can be defined on a job to prevent simultaneous actions on the same environment
 - Example – You don’t want two jobs deploying code to same environment concurrently

Example Deployment Jobs

- “[environment](#)” is either “production” or “staging”
 - Based on what branch pipeline running on
- “[resource_group](#)” prevents two jobs using same environment from running at same time
 - Jobs can be in same pipeline or in different ones

```
deploy-production-host:
  extends: .tf-deploy-host
  variables:
    ENVIRONMENT_NAME: production
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

deploy-staging-host:
  extends: .tf-deploy-host
  variables:
    ENVIRONMENT_NAME: staging
  rules:
    - if: $CI_COMMIT_BRANCH == "staging"

.tf-deploy-host:
  variables:
    TF_VAR_ENVIRONMENT: ${ENVIRONMENT_NAME}
    TF_STATE_NAME: ${ENVIRONMENT_NAME}
  environment:
    name: ${ENVIRONMENT_NAME}
  resource_group: ${ENVIRONMENT_NAME}
  script:
    - terraform apply -input=false \
      -target="module.ec2_instance" \
      -target="module.security_group" \
      -target="module.vpc" \
      "${TF_PLAN_CACHE}"
```

Note: Example jobs simplified for slide.



Menu

demo-python-tacacs

Project information

Repository

Issues0

Merge requests0

CI/CD

Security & Compliance

Deployments

Feature Flags

Environments

Releases

Packages & Registries

Infrastructure

Monitor

Analytics

Wiki

Snippets

Settings

Hank > demo-python-tacacs > Environments

Available2Stopped0

Enable review appNew environment

production

Stop

SuccessLatest Deployed#5315ac5c864 days ago

Hide details

Merge branch 'staging' into 'main' Update main with CICD changes See merge request hpreston/demo-python-tacacs!2

TriggererJobBranch

@hprestondestroy...tion-hostmain

staging

Stop

SuccessLatest Deployed#7427c584c958 minutes ago

move docker login to before script

TriggererJobBranch

@hprestondeploy-...ng-dockerstaging

Hank > demo-python-tacacs > Jobs > #2530581939

passedJob deploy-staging-docker triggered 54 minutes ago by Hank

This job is deployed to staging.

Running with gitlab-runner 14.10.0~beta.50.g1f2fe53e (1f2fe53e)
on blue-3.shared.runners-manager.gitlab.com/default zwxgkjAP

Preparing the "docker+machine" executor00:09

Using Docker executor with image registry.gitlab.com/gitlab-org/terraform-images/releases/terraform:1.1.9 ...
Authenticating with credentials from job payload (GitLab Registry)
Pulling docker image registry.gitlab.com/gitlab-org/terraform-images/releases/terraform:1.1.9 ...
Using docker image sha256:57e1f2a7b728cd3dbc3cd64583ebf563445bc5a05f7e76c3aa9b8463367d3aa3 for registry.gitlab.com/gitlab-org/terraform-images/releases/terraform:1.1.9 with digest registry.gitlab.com/gitlab-org/terraform-images/releases/terraform@sha256:41e0396658a5c3b87bf0958d649f1b22f4150fc246fbee7efa18de95489b897e ...

Preparing environment00:01

Running on runner-zwxgkjap-project-34435628-concurrent-0 via runner-zwxgkjap-shared-1654047773-8cba46f3...

A dash of reality about CICD

You can't automate
what you don't
understand very
very very well.



The path isn't
always $A \rightarrow B \rightarrow C$

1. Visit Disney World
2. Ride Space Mountain
3. Meet Mickey Mouse

1. Travel to Florida
2. Visit Disney World
3. Eat Mickey Mouse Pretzel
4. Ride Space Mountain
5. Get picture with Mickey Mouse
6. Watch Fireworks

1. Book plane tickets
2. Book hotel
3. Book park tickets
4. Travel to Florida
5. Check in to Hotel
6. Check tram schedule
7. Arrive early to get in line
8. Ride Space Mountain
9. Ride Splash Mountain
10. Get picture with Goofy
11. Watch Fireworks

Do you just
automate the
complexity?



All Done!

Hank Preston

- Example Code: [hpreston/demo-python-tacacs](https://github.com/hpreston/demo-python-tacacs)
- [Full CI/CD Pipeline Demo Video](#)
- Documentation: <https://docs.gitlab.com/ee/ci/>
- Stay in touch:
 - Webex/Email: hapresto@cisco.com
 - Blogs: <https://blogs.cisco.com/author/hankpreston>



CISCO *Live!*

Did you know?

You can have a
one-on-one session with
a technical expert!

Visit Meet the Expert in The HUB
to meet, greet, whiteboard & gain
insights about your unique questions
with the best of the best.



Meet the Expert Opening Hours:

Tuesday	3:00pm – 7:00pm
Wednesday	11:15am – 7:00pm
Thursday	9:30am – 4:00pm
Friday	10:30am – 1:30pm

Session Surveys

We would love to know your feedback on this session!

- Complete a minimum of four session surveys and the overall event surveys to claim a Cisco Live T-Shirt



Continue your education



- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Expert meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand



The bridge to possible

Thank you

CISCO *Live!*

#CiscoLiveAPJC

The background features a vibrant, multi-colored abstract design. On the left, there are horizontal, wavy bands of color in shades of red, orange, yellow, and green. On the right, a bright white light source emits a series of colorful rays in shades of blue, green, and yellow, creating a sunburst effect.

CISCO *Live!*

Let's go

#CiscoLiveAPJC