The bridge to possible

Realising DevOps Through Cisco Technologies

The Story

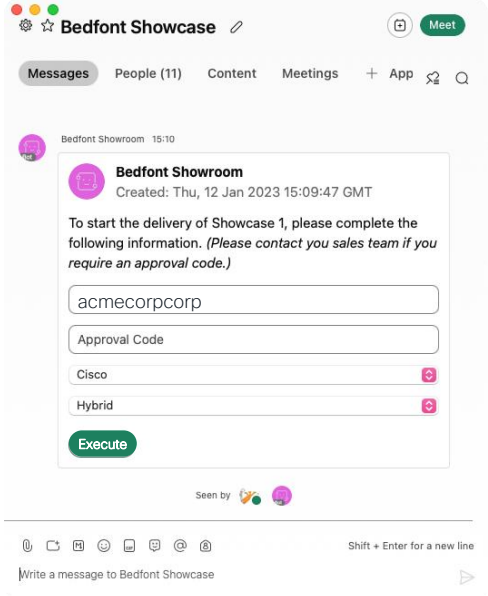
Neil A Trevains, Rob Porter, and Russ Whitear
Solutions Architects, UKI Cloud Infrastructure and Software Group

Agenda

- Audience Participation!
- Automation is <u>not</u> ‘Auto-Magic’!
- The Showcase Pipeline
- What just happened??!!
- Next Steps

Audience Participation…

It’s On The Cards…

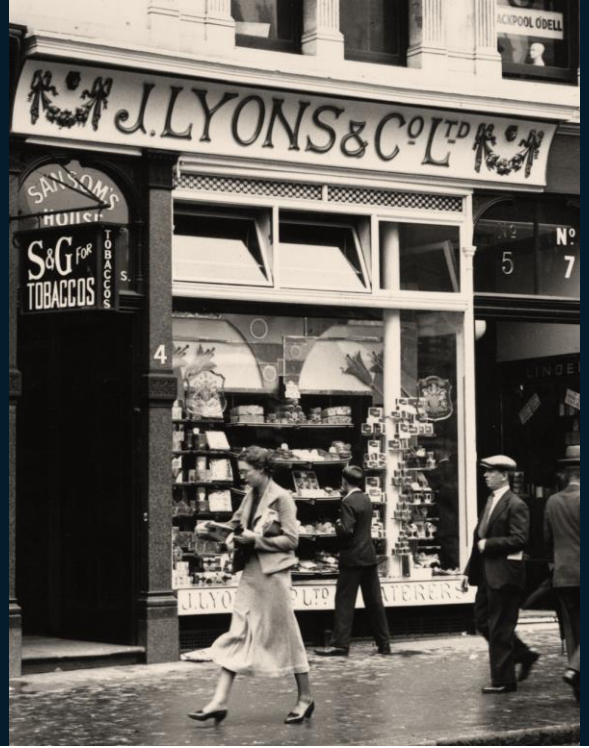


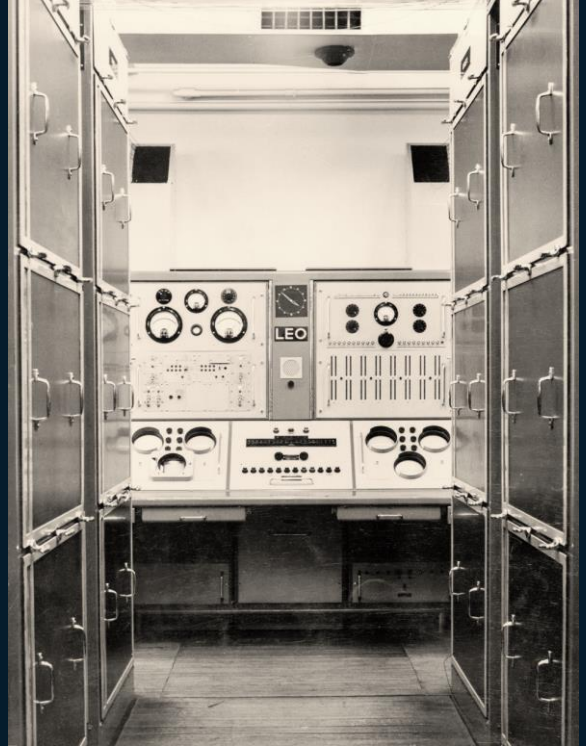
… We'll come back to this later…

"Each business is a victim of Digital Darwinism, the evolution of consumer behavior when society and technology evolve faster than the ability to exploit it. Digital Darwinism does not discriminate. Every business is threatened."

Brian Solis, Principal Analyst

Altimeter, a Prophet Company





Is the pursuit of ‘Digital Transformation’ really new…?

The Digital Supply Chain

Hybrid Cloud = Complexity^{n}

The ‘New’ Normal…

NIST Cloud – Essential Characteristics

- On-demand Self Service
- Broad Network Access
- Resource Pooling
- Rapid Elasticity
- Measured Service

On-premises DC

Edge

Hybrid IT

Co-Lo DC

Multiple Clouds

Bridge the ’Agility Gap’
via **DC Modernisation**

On-premises DC

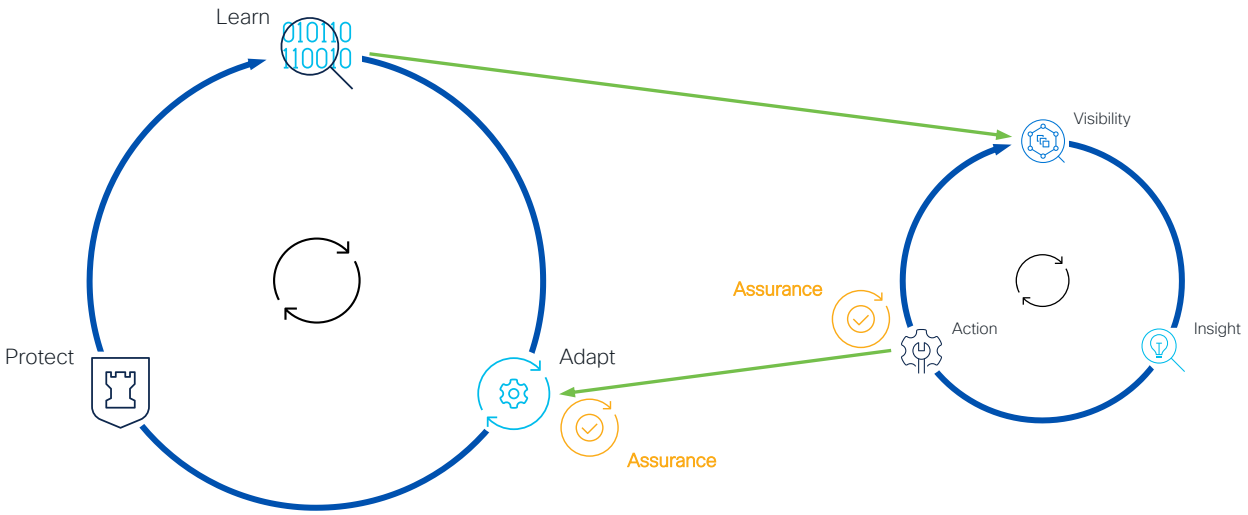
Public Cloud

Observability

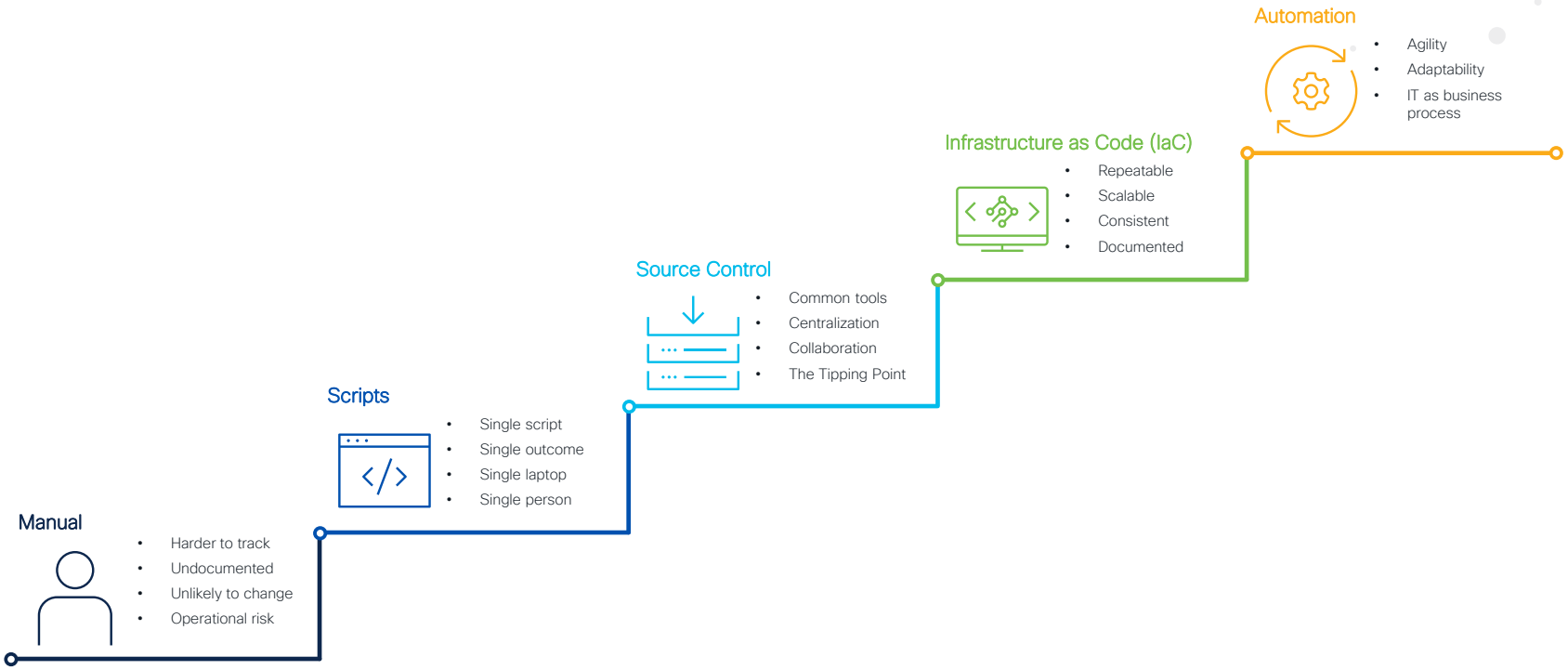
Machine Intelligence

Pervasive Automation

Modernization: Insights, Automation, Assurance



Evolution of Infrastructure Configuration



IT Resilience for the Digital Age

Seven core beliefs on what it takes to achieve resiliency

Solve for journeys, not applications

Take a risk-based approach

Leverage IT operations data

Design for the storm, not the blue skies

Adopt an engineering mindset

Avoid hero culture

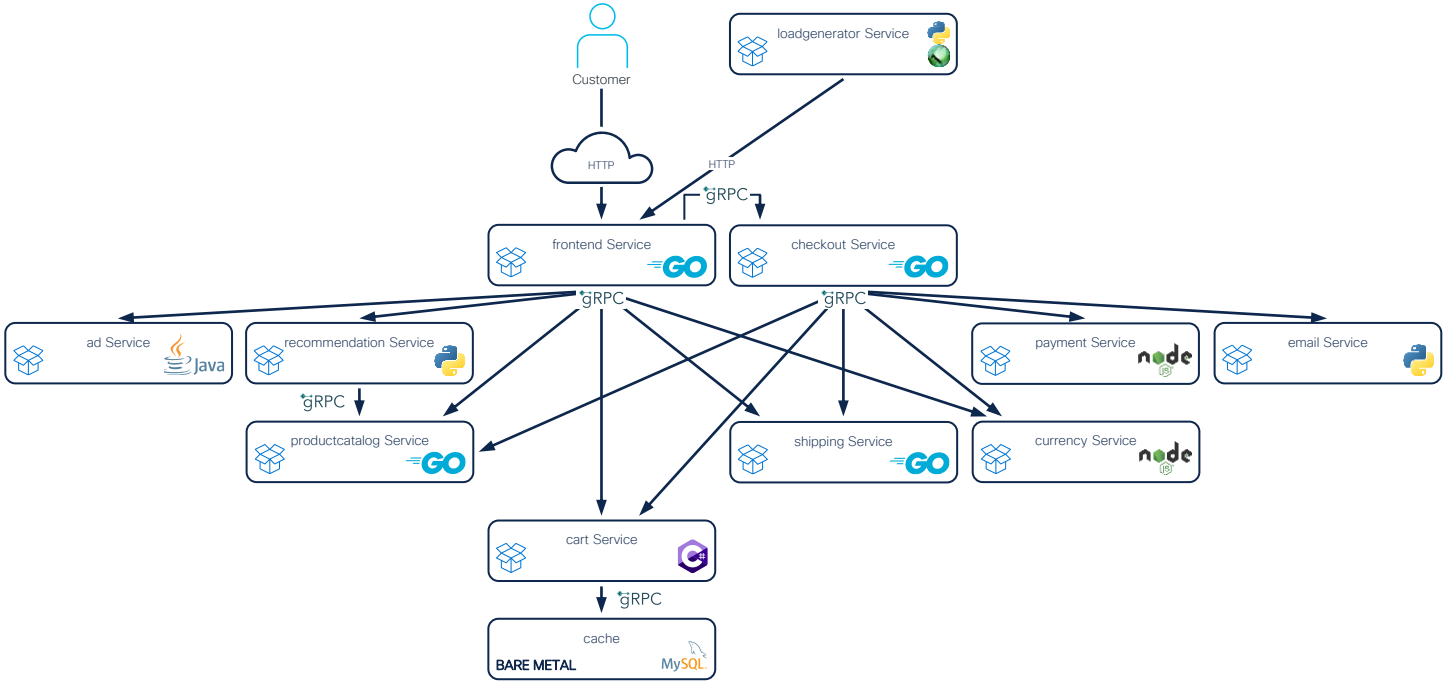
Become proactive, not reactive

Source: McKinsey Digital - 11 May 2021- by Arun Gundurao, Jorge Machado, Rut Patel, and Yanwing Wong

Now, why did we pick a card, and why is the name significant…?

Google Microservices Demo Application

Online Boutique



Showcase Pipeline

Stage 1/1a — Validate
- Input
- Validate Terraform

Stage 2 — Preflight
- Capacity check

Stage 3 — Test-infra
- Test infrastructure

Stage 4 — Plan
- Terraform plan

Stage 5 — Deploy

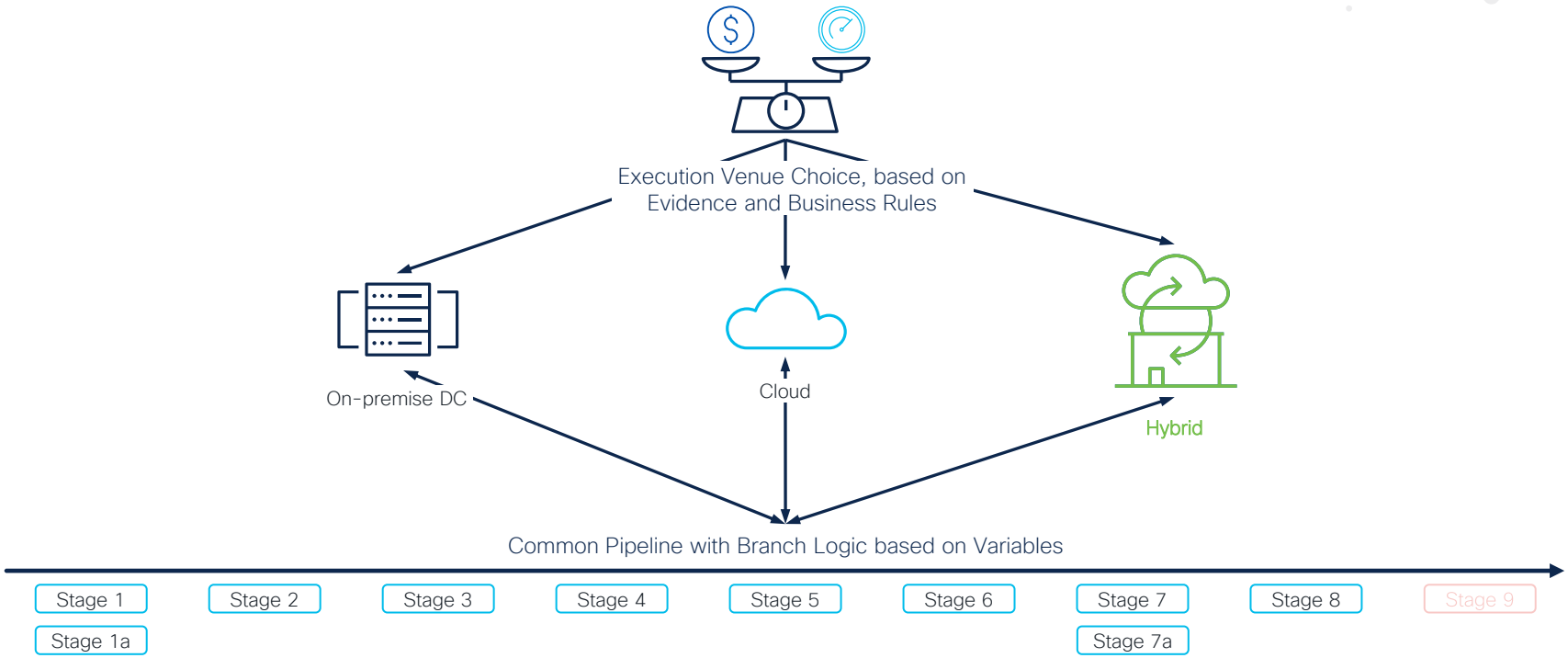
Choice:
- On-premise
- Cloud
- Hybrid

Stage 6 — Compliance
- 1000Eyes

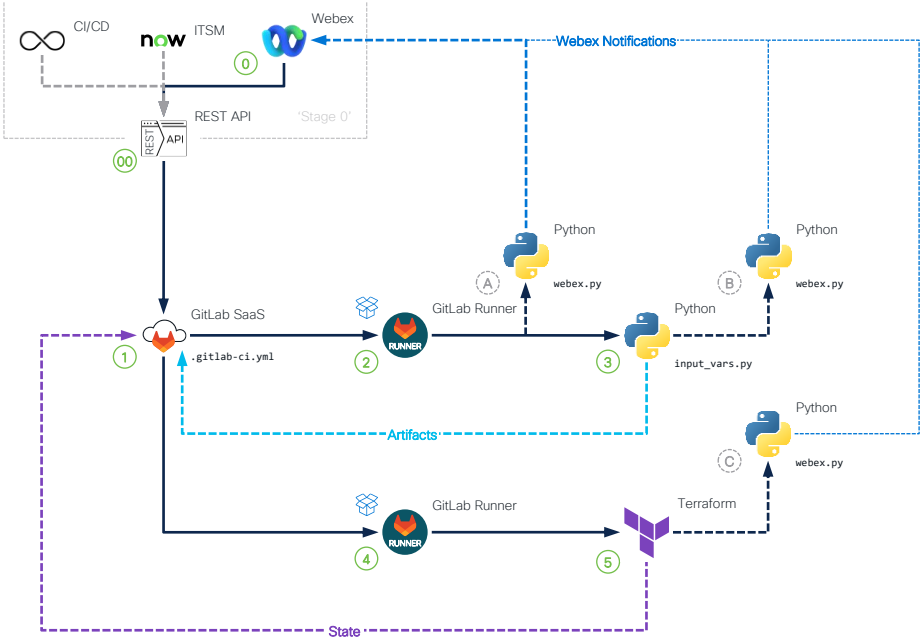
Stage 7/7a — Test
- GUI
- Test VMs

Stage 8 — Webex
- Notify

Execution Venue Choice, Common Pipeline



Showcase Pipeline – Stages 1 & 1a – Validate



0 Initial request – integration with existing CI/CD development pipeline, or selection of a service catalog item, or a ChatBot (e.g., **Webex bot**).

00 Regardless of the Stage 0 method, the result is a **REST API** call to **GitLab SaaS** in order to trigger the pipeline.

1 On receiving the trigger, GitLab begins to process the pipeline configuration file (`.gitlab-ci.yml`) within the repo.

2 4 **GitLab SaaS** has no access to the on-premise estate and must communicate with an on-premise **GitLab Runner** agent.

3 `input_vars.py` carries out two checks on the company name that was entered in the **Webex bot** card:

- GitLab CI/CD variable – can only contain letters (a-z), numbers (0-9), and underscore (_)
- IETF RFC952 – hostname can only contain letters (a-z), numbers (0-9), dash (-), and period (.)

The application services must be accessible via their FQDNs.

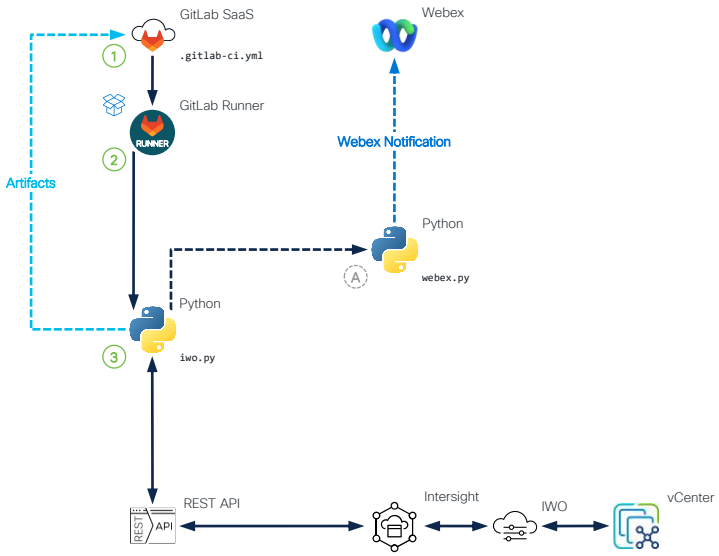
5 A *terraform validate* command is issued. This validates the **Terraform** configuration files within the repo. It checks to ensure that the configuration is syntactically valid and internally consistent, regardless of any provided variables, existing state, provider APIs, etc.

A B C (Optional) `webex.py` returns a message back to the **Webex** card, either a successful job within a stage or a successful stage.

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

Showcase Pipeline – Stage 2 - Preflight



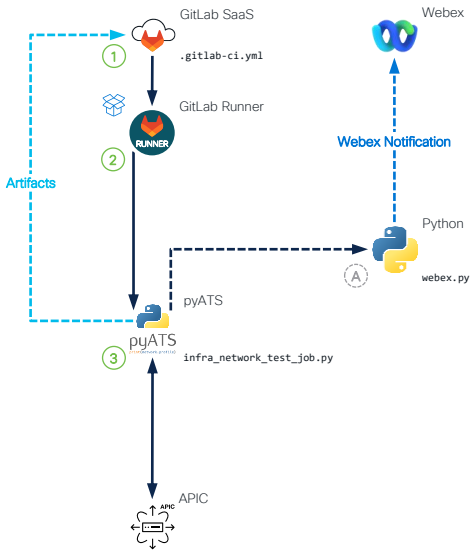
1. The next stage of the pipeline configuration file (`.gitlab-ci.yml`)is processed provided that the previous stage has completed successfully.
2. As in the previous stage, the on-premise **GitLab Runner** agent receives a job request from **GitLab SaaS** and creates a **Docker** executor.
3. `iwo.py` uses a **REST API** call to query **Intersight Workload Optimizer (IWO)** within **Intersight**. IWO confirms whether there is adequate capacity for four virtual machines that will be created on-premise, based on information from a previously claimed **vCenter**. If capacity is available, it will be reserved.

A (Optional) `webex.py` returns a message back to the **Webex** card, either a successful job within a stage or a successful stage.

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

Showcase Pipeline – Stage 3 – Test-infra



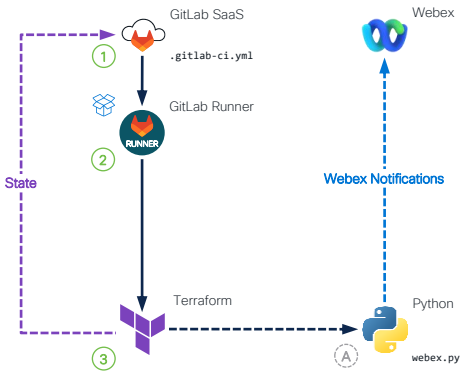
1. The next stage of the pipeline configuration file (`.gitlab-ci.yml`) is processed provided that the previous stage has completed successfully.
2. As in the previous stage, the on-premise **GitLab Runner** agent receives a job request from **GitLab SaaS** and creates a **Docker** executor.
3. **pyATS** (Cisco's network test automation solution) runs a pre-defined test case (trigger) – `infra_network_test_job.py` – against the on-premise **APIC** cluster to confirm the availability of the following within the target tenant:
   - VRF
   - Bridge Domain

A (Optional) `webex.py` returns a message back to the **Webex** card, either a successful job within a stage or a successful stage.

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

Showcase Pipeline – Stage 4 - Plan



1. The next stage of the pipeline configuration file (`.gitlab-ci.yml`) is processed provided that the previous stage has completed successfully.

2. As in the previous stage, the on-premise **GitLab Runner** agent receives a job request from **GitLab SaaS** and creates a Docker executor.

3. A *terraform plan* command is issued. **Terraform** carries out the following:
   - Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.
   - Compares the current configuration to the prior state and notes any differences.
   - Proposes a set of change actions that should, if applied, make the remote objects match the configuration.

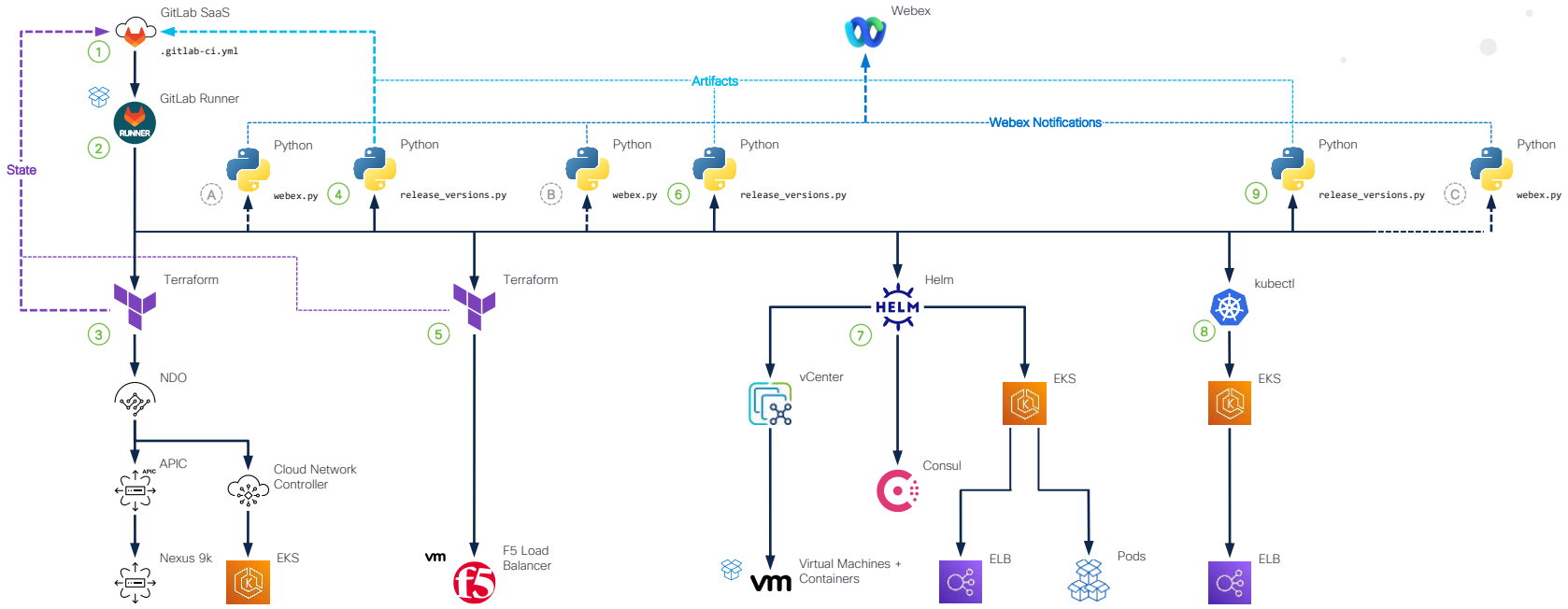
   At this stage, a *speculative plan* exists, which is a description of the effect of the plan without any intent to apply it.

A (Optional) `webex.py` returns a message back to the **Webex** card, either a successful job within a stage or a successful stage.

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

Showcase Pipeline – Stage 5 - Deploy



Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

Showcase Pipeline – Stage 5 - Deploy

1. The next stage of the pipeline configuration file (`.gitlab-ci.yml`) is processed provided that the previous stage has completed successfully.

2. As in the previous stage, the on-premise **GitLab Runner** agent receives a job request from **GitLab SaaS** and creates a **Docker** executor.

3. **Terraform** provisions the required network constructs via **Nexus Dashboard Orchestrator**, creating Application Network Profiles (ANPs), End-Point Groups (EPGs), and contracts for the front-end, middleware, and back-end, both on-premise and in AWS.

4 6 9. `release_versions.py` creates an artifact file containing key information require in subsequent steps of the stage and the pipeline as a whole.

5. **Terraform** creates a new virtual IP, and an associated rule within the on-premise **F5 Load Balancer VM** for external communication with the **AWS EKS** installed application components.

7. **Helm** then deploys the application components:
   - Four as containers embedded in VMs
   - Seven as containers within an **AWS EKS** instance

   The on-premise **Consul** service mesh is reconfigured to host a domain specific to the current deployment ID.

   Within the **AWS EKS** instance, two configuration tasks are carried out:
   - **CoreDNS** is given the on-premise **Consul** as it's next-hop authoritative server
   - An **ELB** is created to provide ingress to the **EKS** environment

8. **kubectl** (within a **bash shell**) then queries the **ELB** and returns the raw URL of the application front-end, to be stored as a variable.

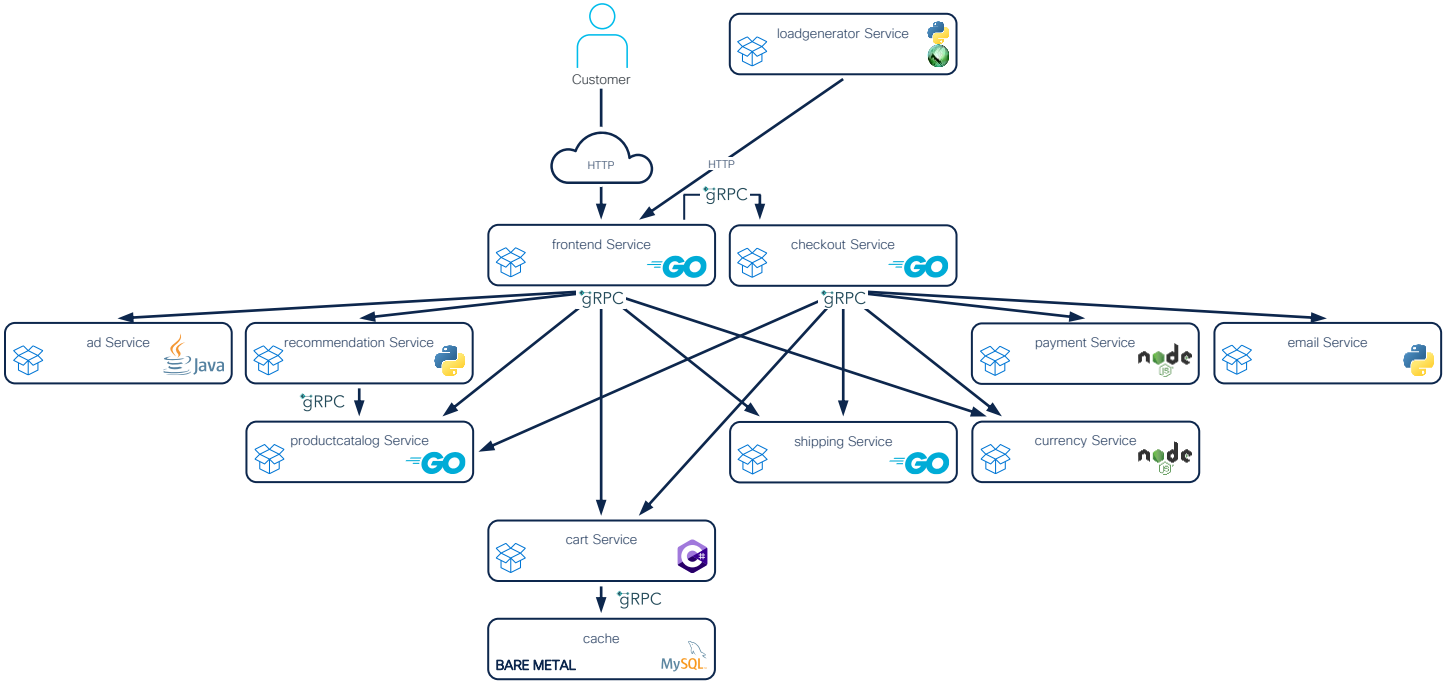
A B C (Optional) `webex.py` returns a message back to the **Webex** card, either a successful job within a stage or a successful stage.

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

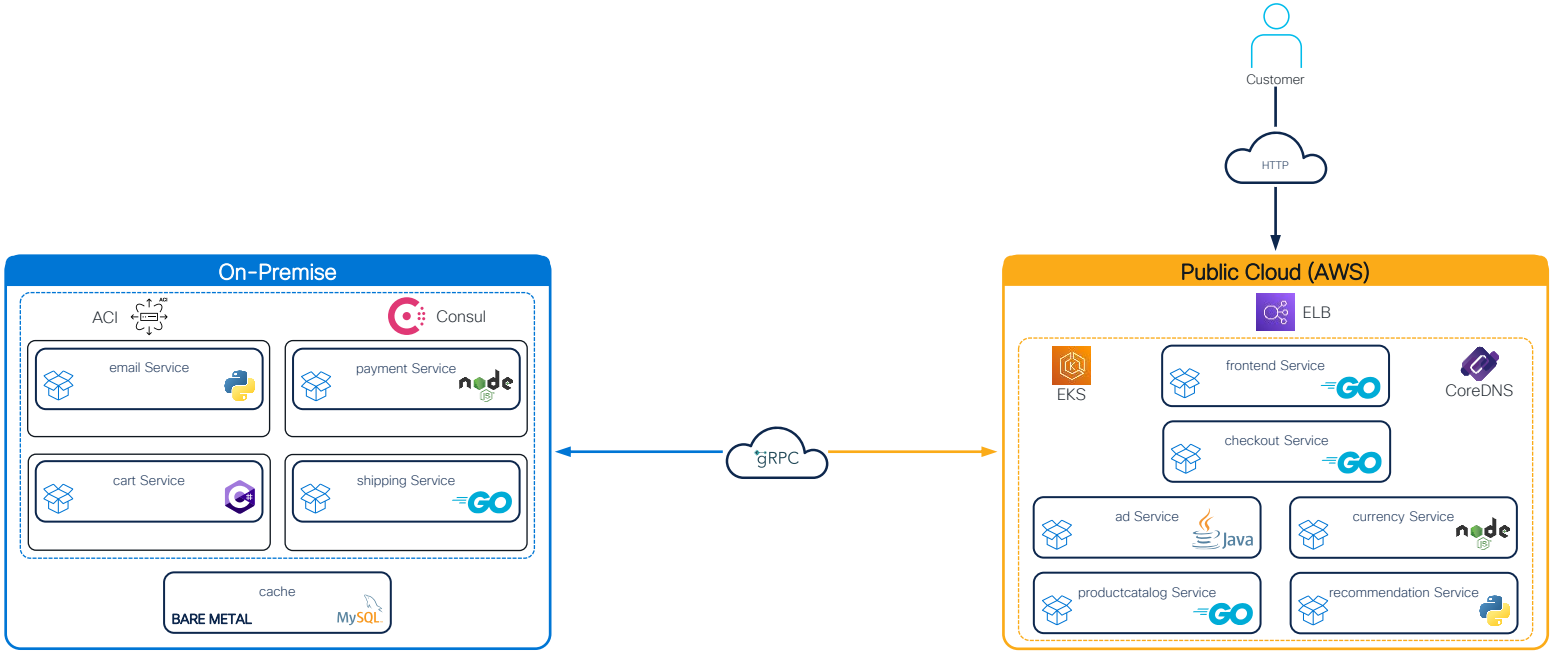
Showcase Pipeline – Stage 5 - Deploy

Google Microservices Demo Application – Online Boutique

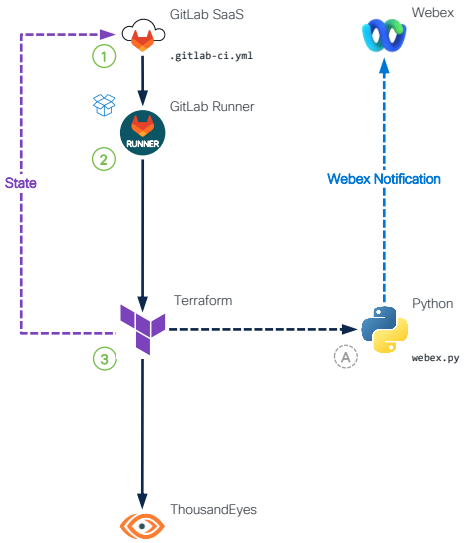


Showcase Pipeline – Stage 5 - Deploy

Google Microservices Demo Application – Online Boutique – As Deployed



Showcase Pipeline – Stage 6 - Compliance



1. The next stage of the pipeline configuration file (`.gitlab-ci.yml`) is processed provided that the previous stage has completed successfully.
2. As in the previous stage, the on-premise **GitLab Runner** agent receives a job request from **GitLab SaaS** and creates a Docker executor.
3. A *terraform plan* command is issued. This creates a `plan.cache` file which is subsequently used by a *terraform apply* command. This deploys a new **ThousandEyes** test that checks http accessibility of the URL provided by the AWS ELB from two locations:
   - An on-premise agent
   - A standard **ThousandEyes** agent installed in the same AWS region used by the **ELB**

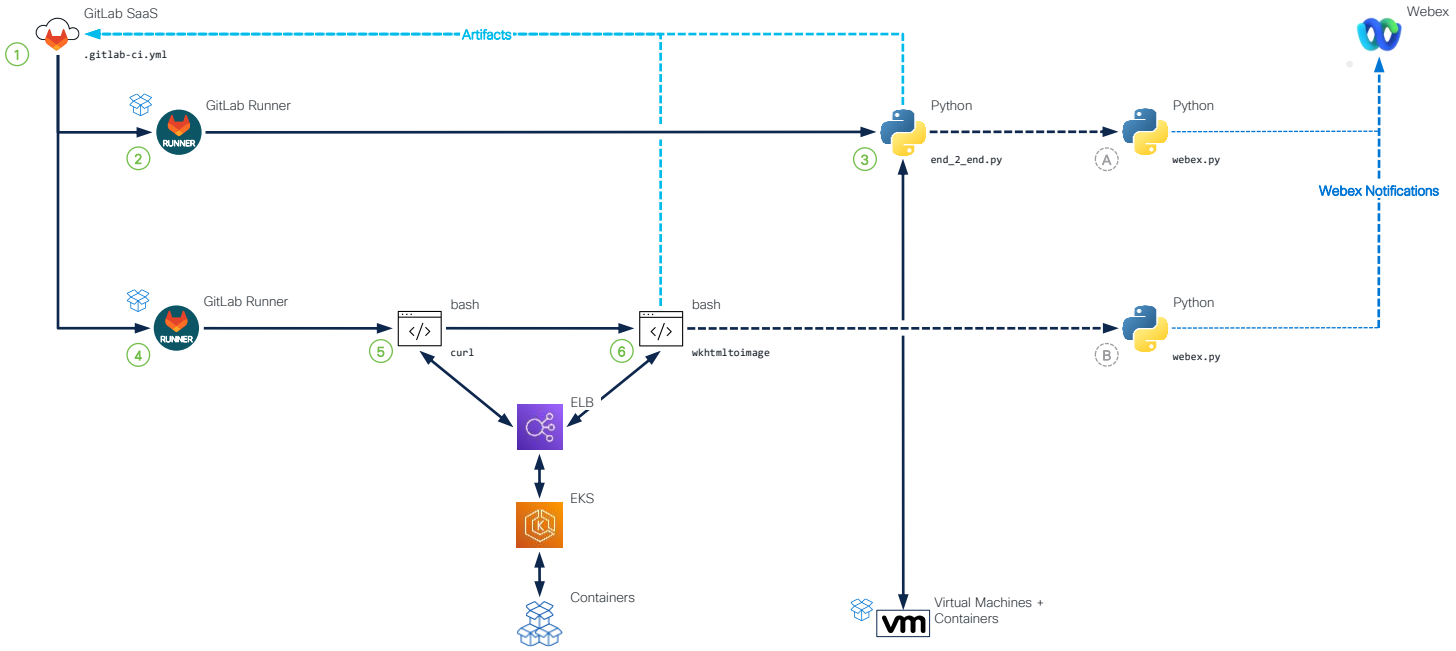
   If this test fails, a **PagerDuty** alert is issued.

A. (Optional) `webex.py` returns a message back to the **Webex** card, either a successful job within a stage or a successful stage.

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

Showcase Pipeline – Stages 7 & 7a - Test



Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

Showcase Pipeline – Stages 7 & 7a - Test

1. The next stage of the pipeline configuration file (`.gitlab-ci.yml`) is processed provided that the previous stage has completed successfully.

2, 4. As in the previous stage, the on-premise **GitLab Runner** agent receives a job request from **GitLab SaaS** and creates a Docker executor.

3. Using static, hard-coded port numbers for each service, combined with the previously allocated IP addresses for each of the four on-premise VMs, `end_2_end.py` attempts a low-level socket connection to each service, to confirm that the socket is listening.

5. From a bash shell, the pipeline attempts a **curl** to the raw **ELB** front-end address. If unsuccessful, it waits for 70 seconds and then attempts to connect again.

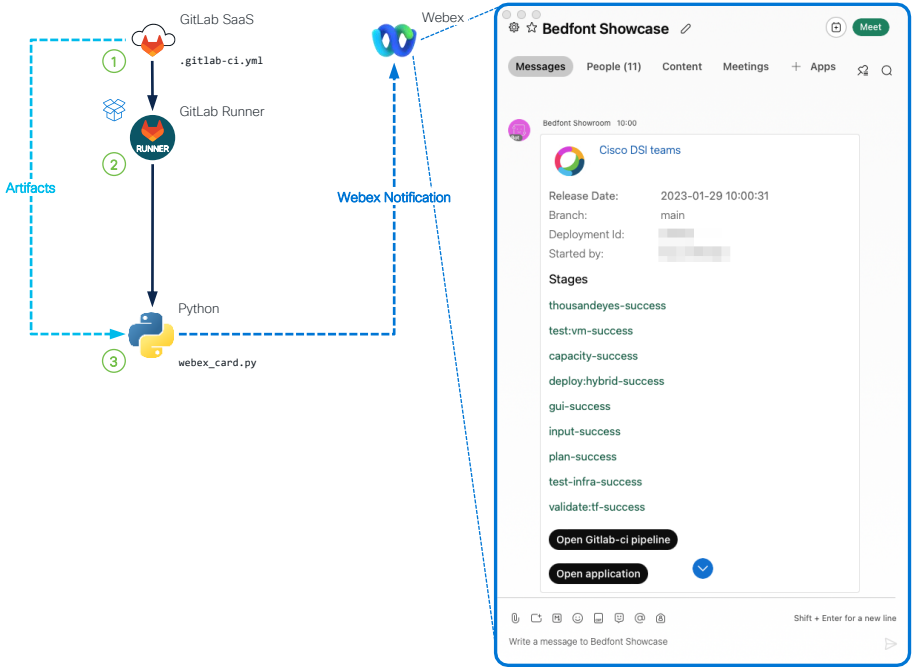
6. Once a connection is successful, `wkhtmltoimage` grabs a screenshot of the front-end page and stores it as an artifact.

A, B. (Optional) `webex.py` returns a message back to the **Webex** card, either a successful job within a stage or a successful stage.

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

Showcase Pipeline – Stage 8 - Webex

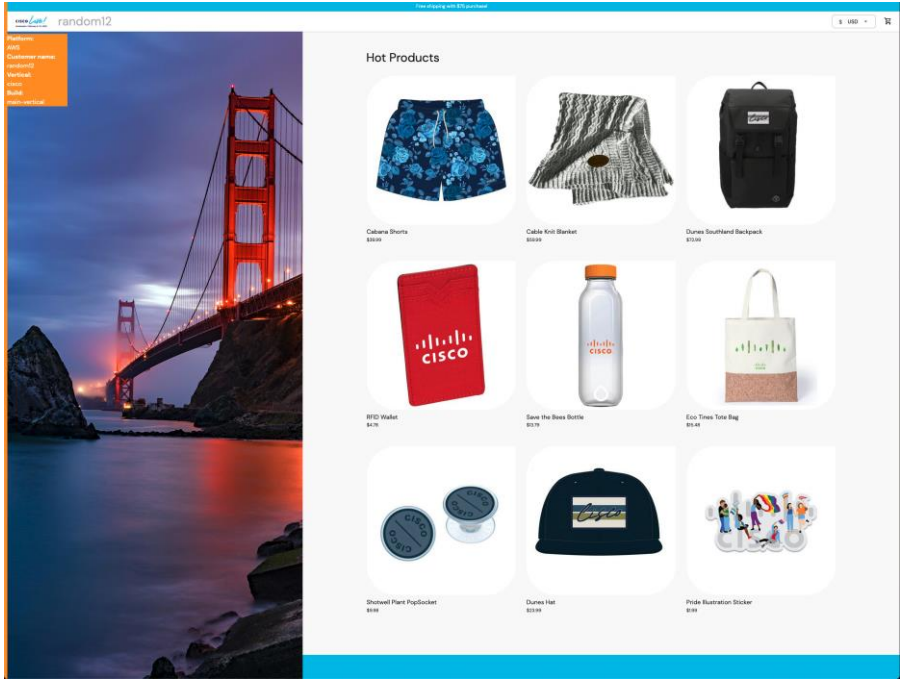


1. The next stage of the pipeline configuration file (`.gitlab-ci.yml`) is processed provided that the previous stage has completed successfully.
2. As in the previous stage, the on-premise **GitLab Runner** agent receives a job request from **GitLab SaaS** and creates a Docker executor.
3. On a successful deployment, `webex_card.py` builds an adaptive card, populated with key variables:
   - Repo branch
   - Deployment ID – the company name entered on initial execution
   - User who requested the build

   Using the artifacts created throughout the pipeline, three further pieces of information are included:
   - Status of each pipeline stage
   - Link to the GitLab CI/CD pipeline for the deployment
   - Link to the front-end of the deployed application

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9
Stage 1a | Stage 7a

The Reveal!

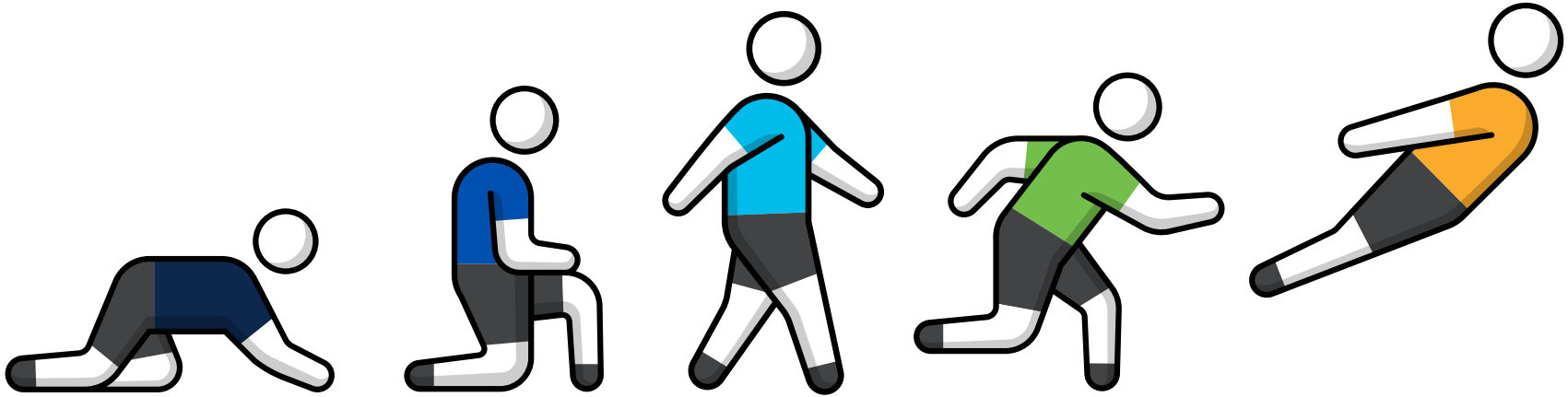


Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7 | Stage 8 | Stage 9

Stage 1a | Stage 7a

That’s great, but we’re a long way from there. Where do we start?

… Crawl, Walk, Run, Fly!



Next Steps

'Crawl, Walk, Run, Fly' should be an **evolution**, not a revolution

Infrastructure as Code: Dynamic Systems for the Cloud Age, 2nd Edition – Kief Morris
Published by O'Reilly, January 2021, ISBN-13 978-1098114671

Cisco DevNet - Learn network programmability basics: https://developer.cisco.com/video/net-prog-basics/

Python – Getting Started: https://www.w3schools.com/python/python_getstarted.asp

HashiCorp Terraform Tutorials - Get Started: https://developer.hashicorp.com/terraform/tutorials

GitLab CI/CD – Getting Started: https://docs.gitlab.com/ee/ci/

Cisco pyATS – Getting Started: https://pubhub.devnetcloud.com/media/pyats/docs/getting_started/index.html

ThousandEyes - Getting Started: https://www.thousandeyes.com/resources/getting-started-tutorial

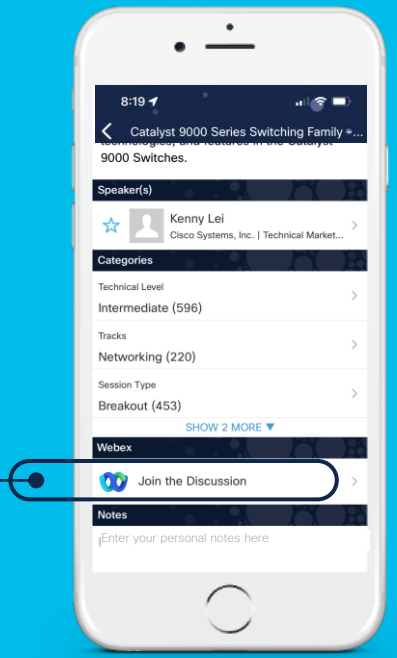
Cisco Webex App

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 until February 24, 2023.

Complete your Session Survey

- Please complete your session survey after each session. Your feedback is very important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (open from Thursday) to receive your Cisco Live t-shirt.

- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Session Catalog and clicking the "Attendee Dashboard" at https://www.ciscolive.com/emea/learn/sessions/session-catalog.html

Continue Your Education

Visit the Cisco Showcase for related demos.

Book your one-on-one Meet the Engineer meeting.

Attend any of the related sessions at the DevNet, Capture the Flag, and Walk-in Labs zones.

Visit the On-Demand Library for more sessions at ciscolive.com/on-demand.

The bridge to possible

Thank you

cisco Live!

CISCO Live!

ALL IN