LIFEOMIC

April 2018

Building a Virtually Air-gapped Secure Environment in AWS

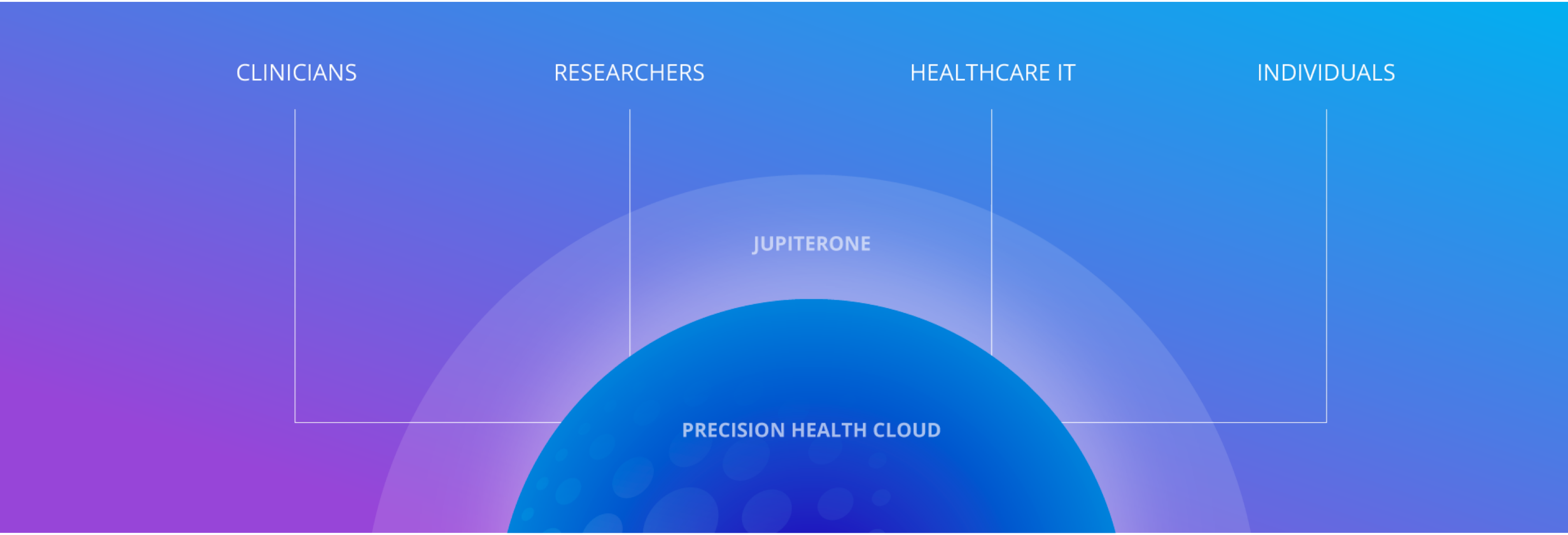
Erkang Zheng
@erkang

Phil Gates-Idem
@philidem

Matt Lavin
@mdlavin

About LifeOmic

Empowering researchers, clinicians and individuals to use data to drive better health outcomes.



Our security challenges

As a technology startup, how do we

- **Allow developers to move fast, work anywhere, feel empowered while ensuring security and compliance?**
- **Prove to auditors and convince customers that their data is safe?**

About this talk

We will cover

1. Forming an effective security program for cloud-native DevSecOps
2. Building a ”virtually air-gapped” production environment in AWS
3. Using a secure software delivery pipeline to promote code into the “air-gapped” environment
4. Automating production change management review and approval (cm-bot)

What this talk is

- Our own security journey
- An opinionated approach
- A selective portion of our security program strategy and technical implementation

What this talk is not

- Not a marketing/sales pitch
- Not a threat landscape view or scientific research
- Not a one-size-fits-all approach or gold standard
- Not a bulletproof cookbook/recipe

1. The Program

The **Assumptions**, **Assurances**, and **Culture** of an effective security program for cloud-native DevSecOps

Our security program journey this past year

2017

APRIL

Assessment and Implementation

- SSO, MFA
- Cloud, server and container monitoring
- Next Gen EPP
- OSS, SAST, DAST, Pen Testing
- Awareness training
- …
- 80+ controls/procedures

DevSecOps Automation and Continuous Improvement

- Hands-free continuous deployment
- Automated security scans
- Automated production change review and approval
- Centralized security analysis and orchestration
- Bug bounty program

JULY

HIPAA Compliance

SEPTEMBER

The “Magnificent Seven”

The “Essential Eight”

The “LifeOmic Top Ten”

Began HITRUST CSF Adoption and Assessment

2018

JANUARY

MARCH

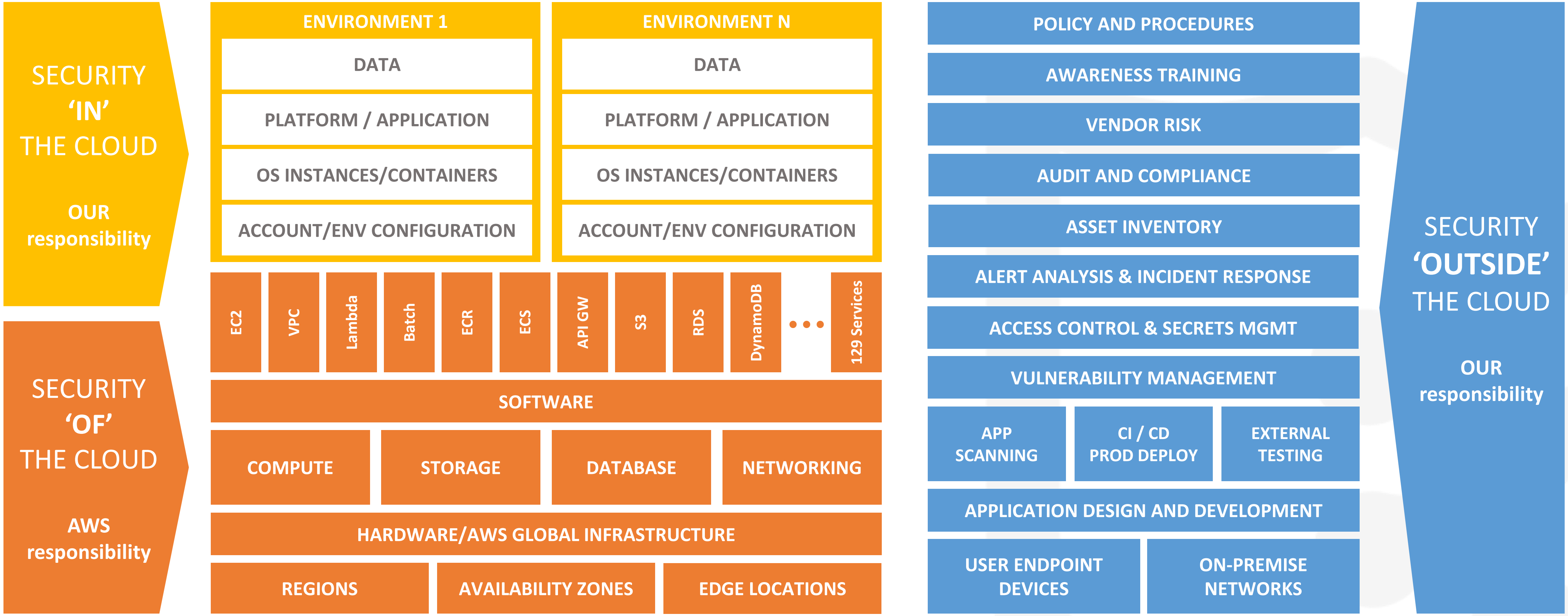
HITRUST CSF Certification

HITRUST CSF Certified

LIFEOMIC

Cloud Security Division of Responsibilities

We assume the security ‘of’ the Cloud can be trusted



SECURITY ‘IN’ THE CLOUD

OUR responsibility

ENVIRONMENT 1

DATA

PLATFORM / APPLICATION

OS INSTANCES/CONTAINERS

ACCOUNT/ENV CONFIGURATION

ENVIRONMENT N

DATA

PLATFORM / APPLICATION

OS INSTANCES/CONTAINERS

ACCOUNT/ENV CONFIGURATION

SECURITY ‘OF’ THE CLOUD

AWS responsibility

EC2 · VPC · Lambda · Batch · ECR · ECS · API GW · S3 · RDS · DynamoDB · • • • · 129 Services

SOFTWARE

COMPUTE · STORAGE · DATABASE · NETWORKING

HARDWARE/AWS GLOBAL INFRASTRUCTURE

REGIONS · AVAILABILITY ZONES · EDGE LOCATIONS

POLICY AND PROCEDURES

AWARENESS TRAINING

VENDOR RISK

AUDIT AND COMPLIANCE

ASSET INVENTORY

ALERT ANALYSIS & INCIDENT RESPONSE

ACCESS CONTROL & SECRETS MGMT

VULNERABILITY MANAGEMENT

APP SCANNING · CI / CD PROD DEPLOY · EXTERNAL TESTING

APPLICATION DESIGN AND DEVELOPMENT

USER ENDPOINT DEVICES · ON-PREMISE NETWORKS

SECURITY ‘OUTSIDE’ THE CLOUD

OUR responsibility

Manifesto

of a cloud native security program

We believe modern cybersecurity, especially for digital companies with cloud-native operations, requires a different mindset and operating model such that we should:

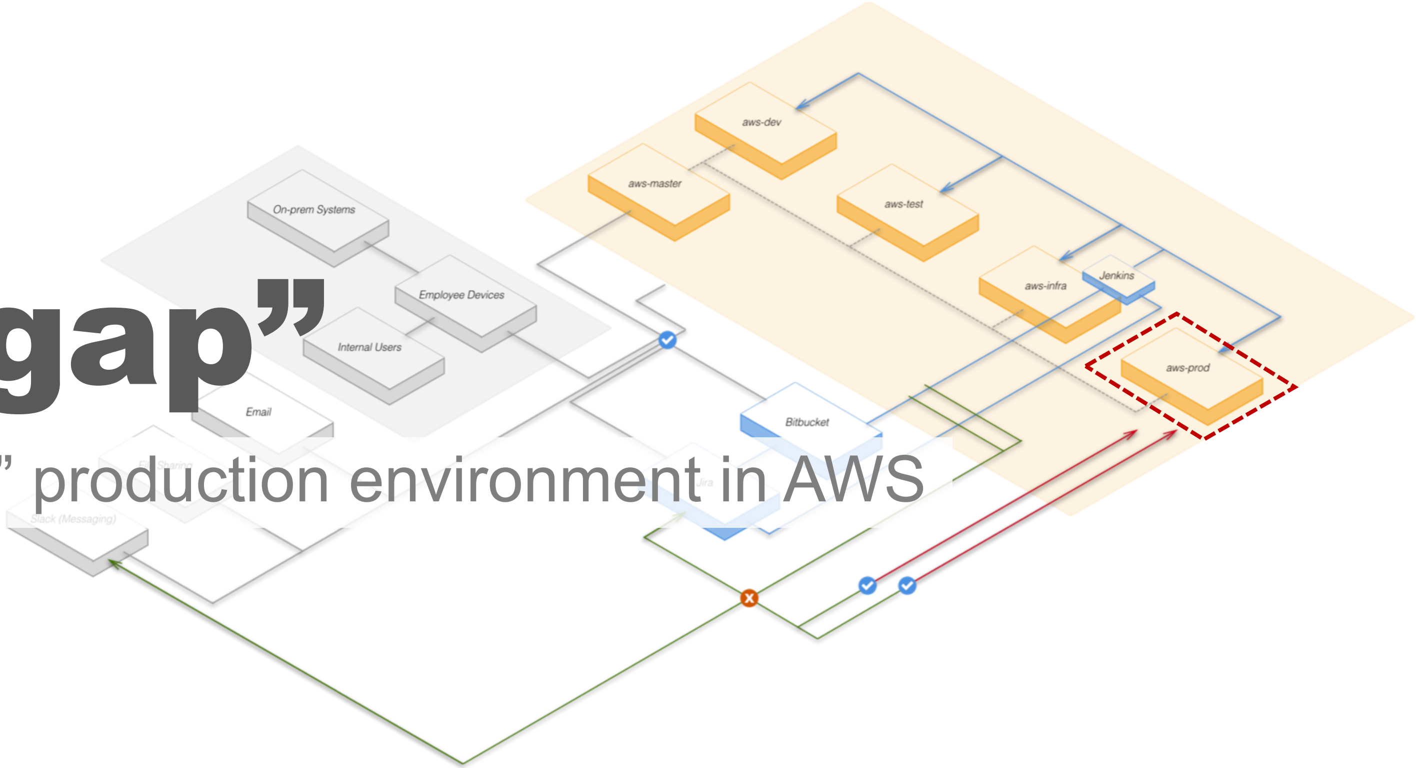
- Assume compromise, but expose no single point of compromise.
- Track everything since you cannot protect what you can't see.
- Automation is key because people don't scale.
- Build products that are secure by design and secure by default.
- Engage everyone in security for there is power in the crowd; two is stronger than one.
- Favor transparency over obscurity, practicality over process, and usability over complexity.

*Security should be **simple**, **open**, **collaborative** and **rewarding**.*

https://securitymanifesto.net

2. The “Air-gap”

Building a ”virtually air-gapped” production environment in AWS



Creating a virtual "air gap" to our production AWS account

The GOALS

For the production environments in AWS, we want to provide the highest level of security assurance, in a way such that

- There is no internal network connectivity into the environment such as VPN, SSH, or AWS DirectConnect.
- Internal engineers can only access applications logs and temporary read-only access in production for troubleshooting and support
- Internal users should have no access to modify systems, configurations, resources, workloads; especially no access to any customer data at all times, even with temporary privileged access

The GATES

Any privileged access into production environment requires an approved changed management ticket and passing four security gates:

- The elevated role must be assigned to the approved individual in the centralized IdP;
- The user must authenticate and pass MFA validation;
- An explicit deny access rule to production must be temporarily lifted for the user to assume a privileged role in production; and
- Even with the privileged access, certain risky actions such as making changes to IAM policies, users, roles or groups and accessing customer data are explicitly denied.

Data-centric model; zero-trust architecture

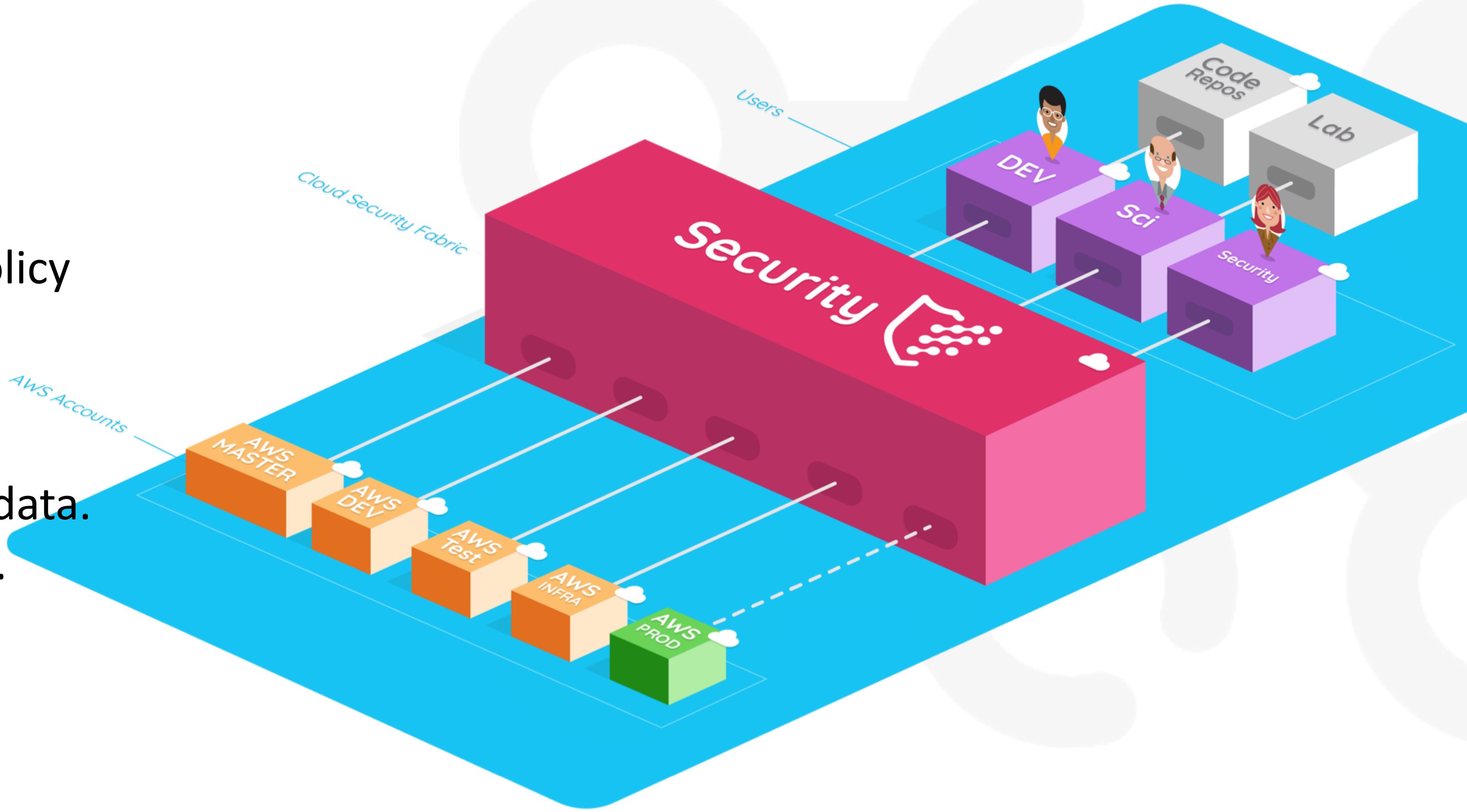
No internal network. 100% cloud.

Fully segregated with Granular policy enforcements.

Individually secured devices.

No internal access to production data. Minimized data leakage potential.

No “keys to the kingdom”;
No single points of compromise.



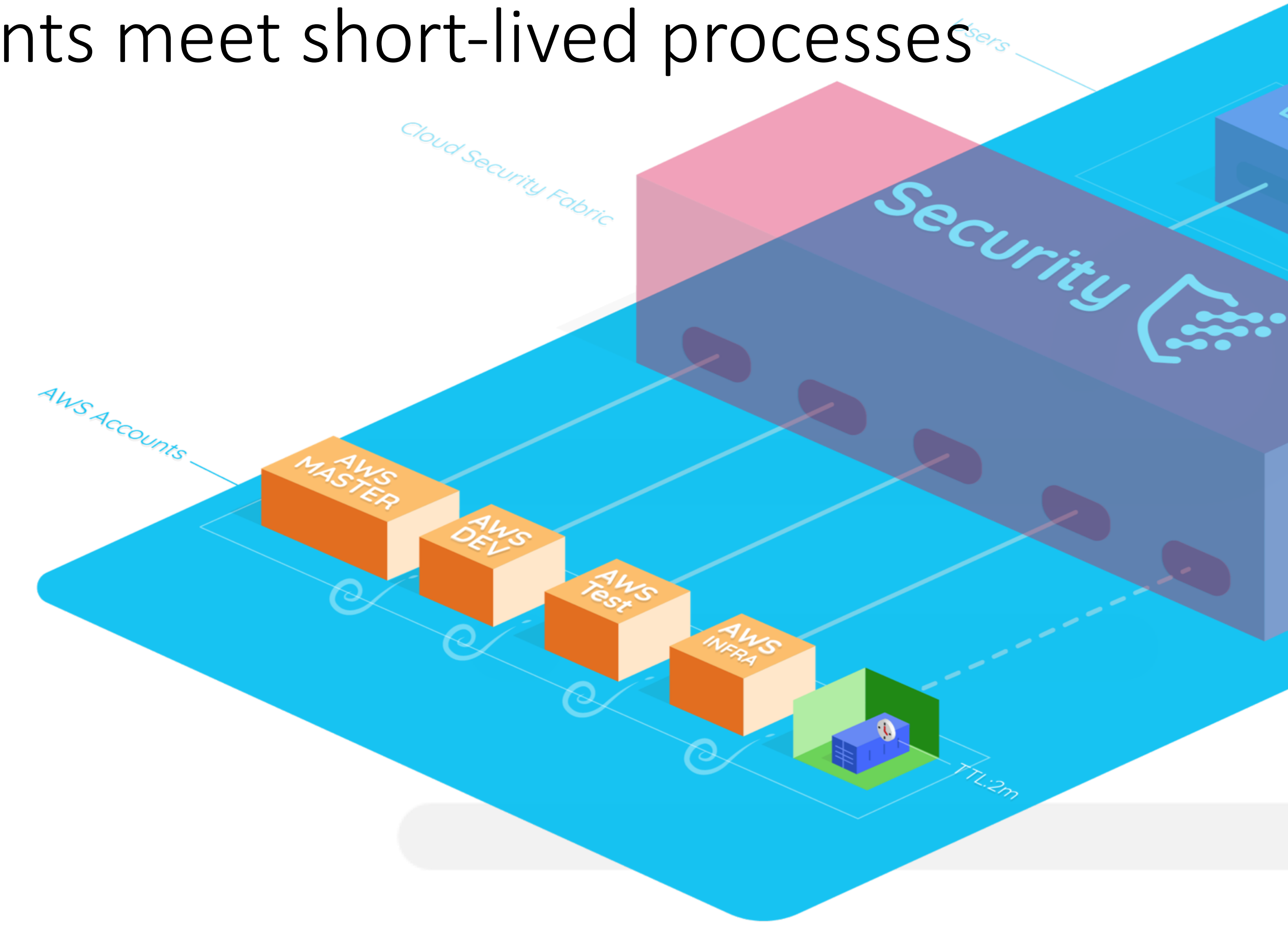
Segregated environments meet short-lived processes

No direct administrative or broad network connectivity into production.

Processes are short-lived and killed after use.

Granular security-group policies.

Minimal persistent attack surface making it virtually impenetrable.



Least-privileged temporary access

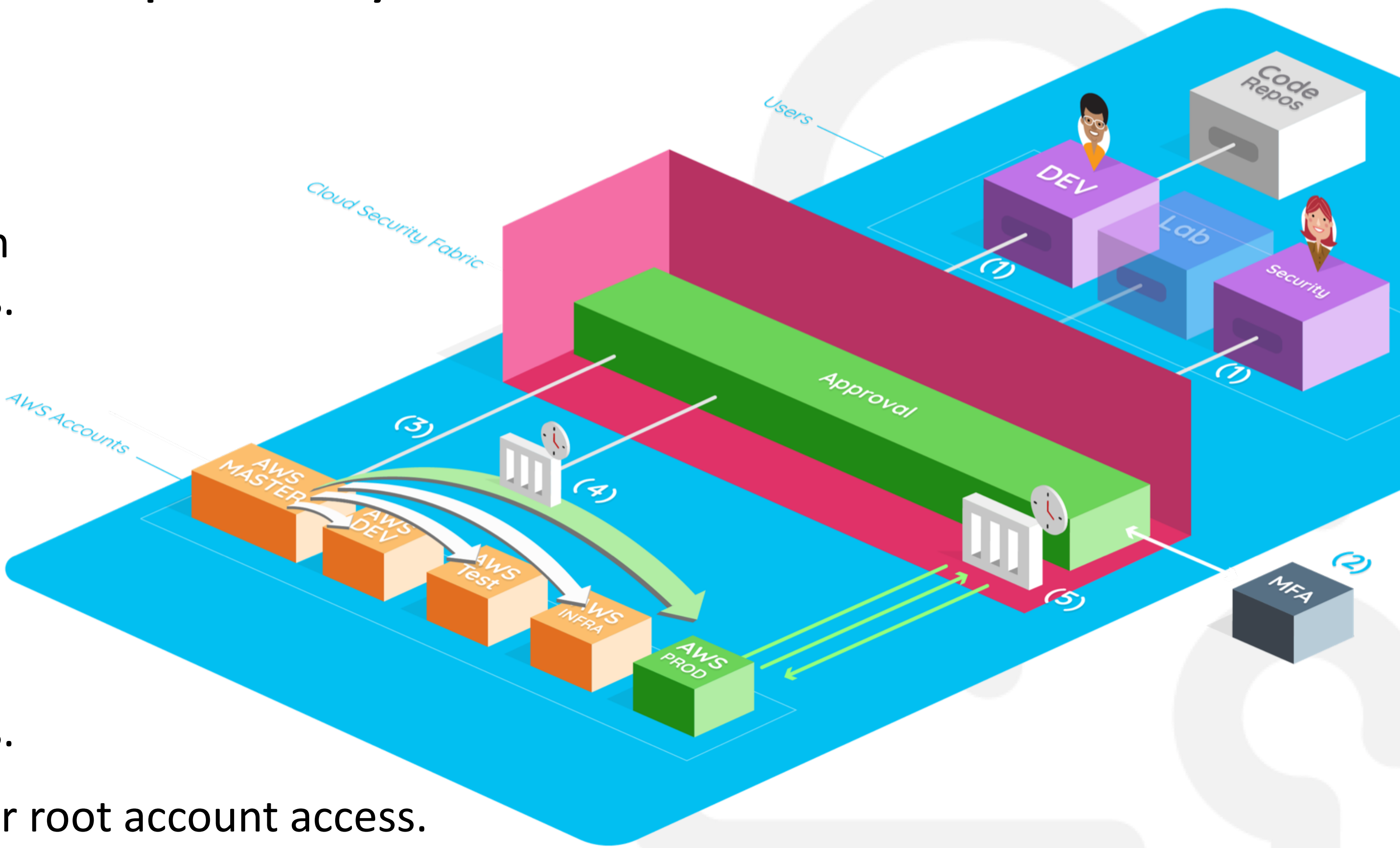
Need-based access control for both employees and computing services.

Access to critical systems and resources are closed by default, granted on demand.

Protected by strong multi-factor authentication.

"Secrets" remain secret at all times.

Split-knowledge and dual-access for root account access.

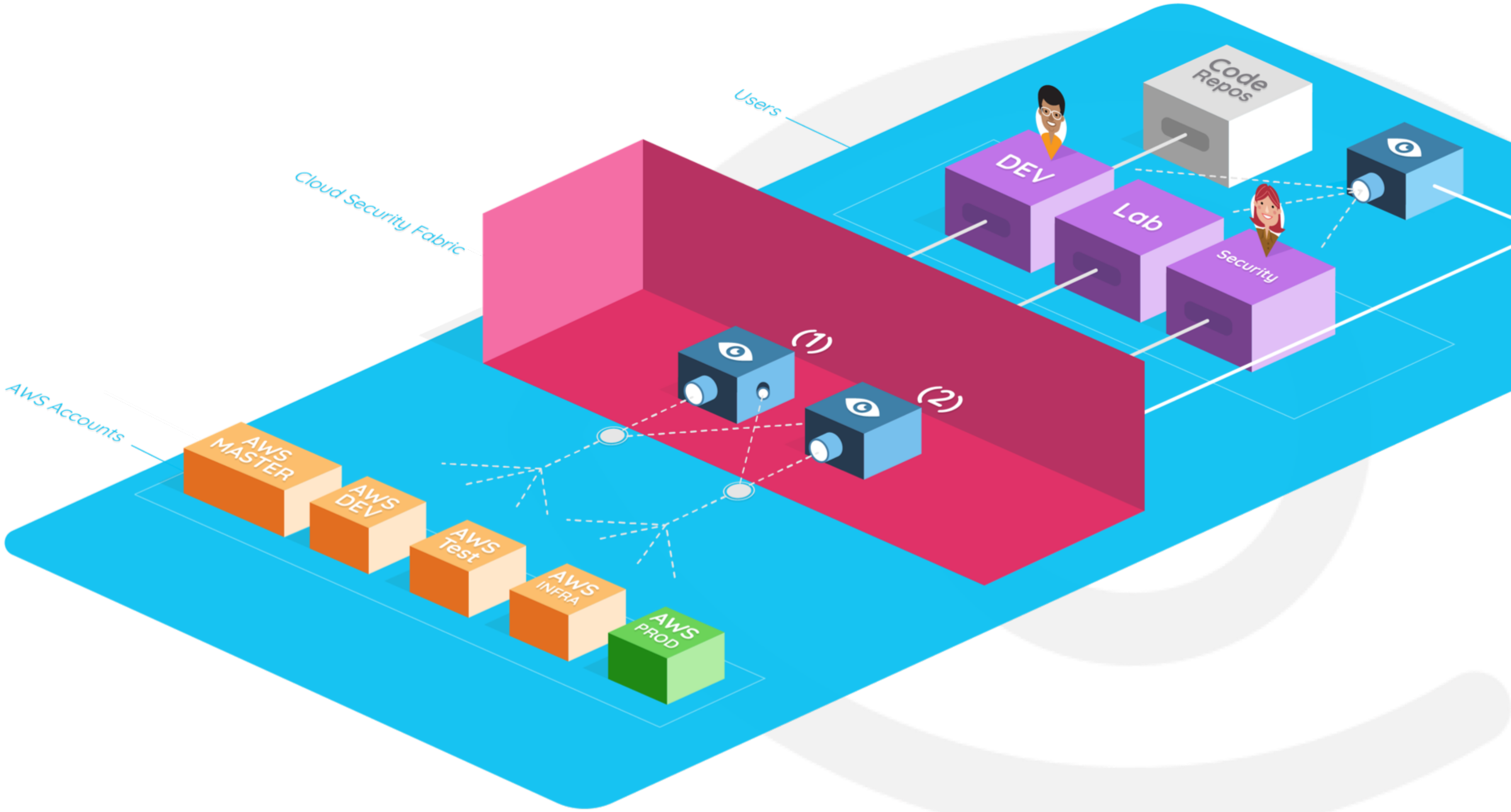


Watch everything, even the watchers

All environments are monitored;
All events are logged;
All alerts are analyzed;
All assets are tracked.

No privileged access without prior approval or full auditing.

We even deployed redundancy to “watch the watchers”.

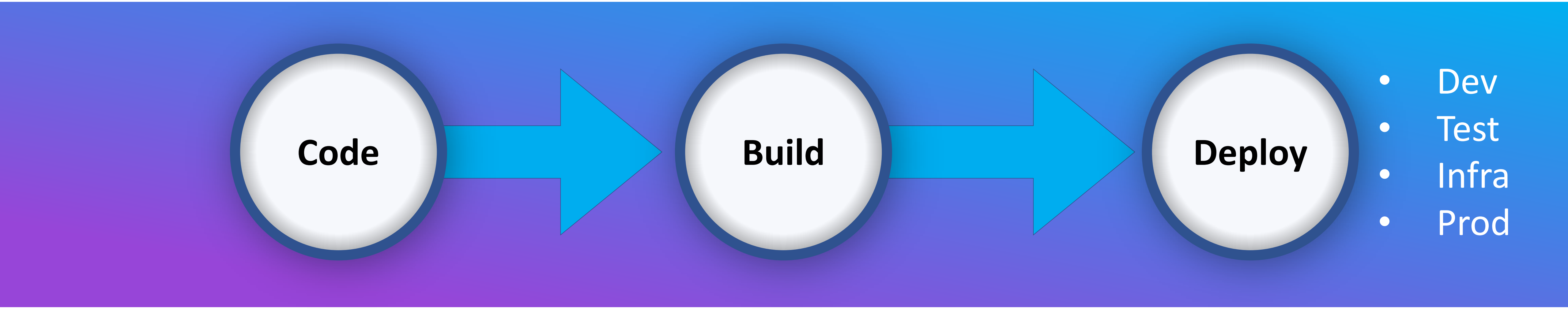


Now, the question is, how do we get software deployed into such an environment without internal network access?

3. The Pipeline

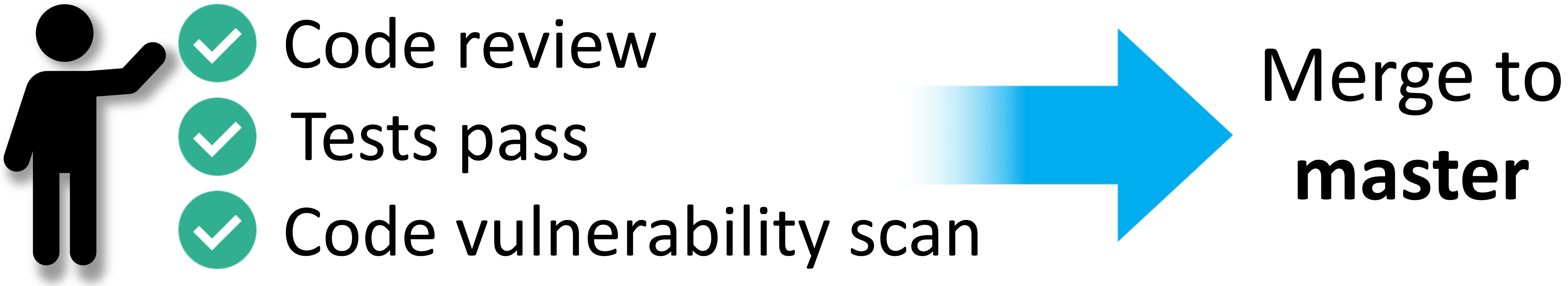
Using a secure software delivery pipeline to promote code into the “air-gapped” environment

The Pipeline Steps



- Dev
- Test
- Infra
- Prod

Code

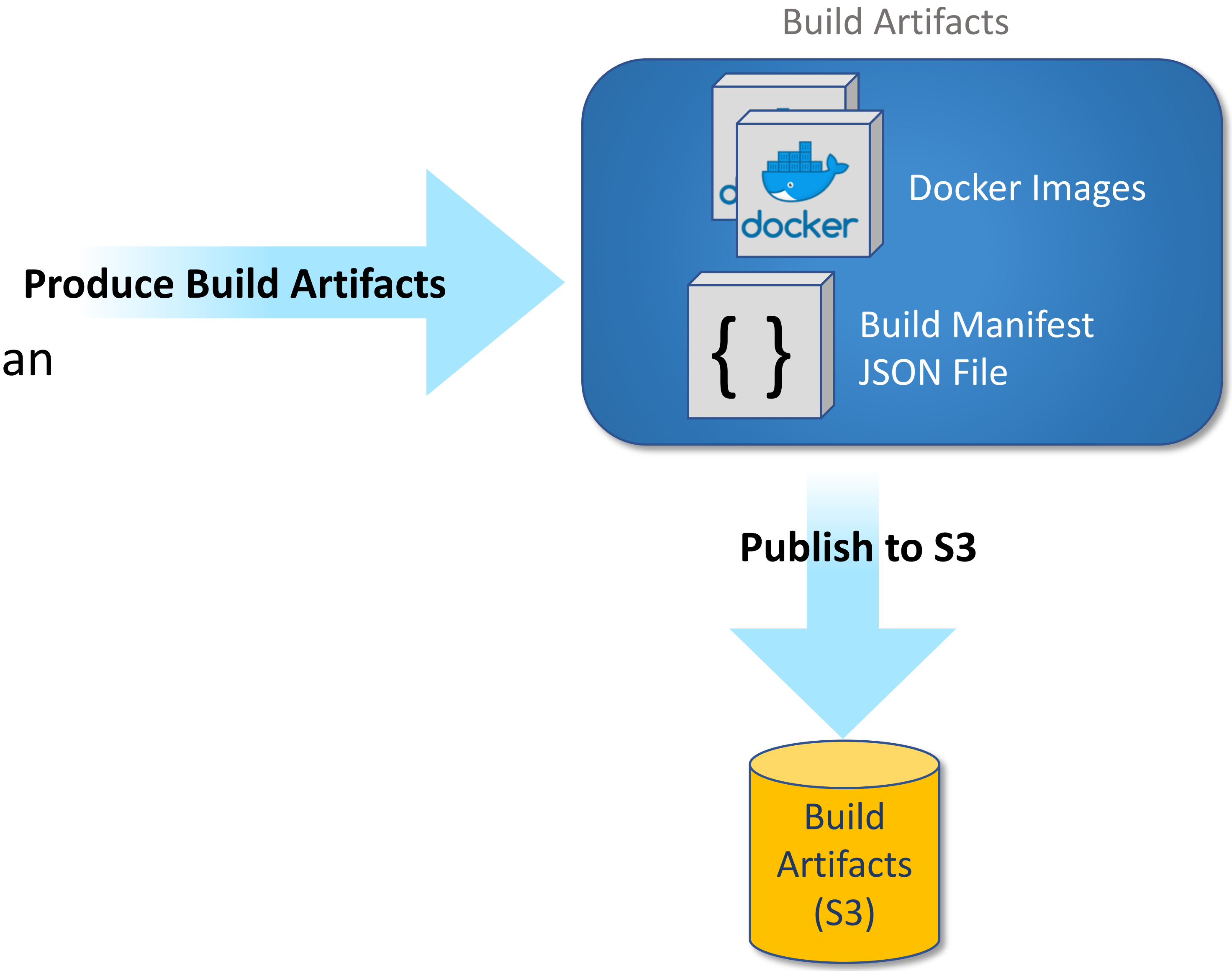


- ✓ Code review
- ✓ Tests pass
- ✓ Code vulnerability scan

Merge to **master**

Build

- ✓ Tests pass
- ✓ Code vulnerability scan



Deploy

Old Way

- VPN network connection between CI/CD service (Jenkins) and target environment
- Changes to infrastructure via UI or shell scripts
- Provisioning via SSH connection
- bastion host / ”jump boxes”

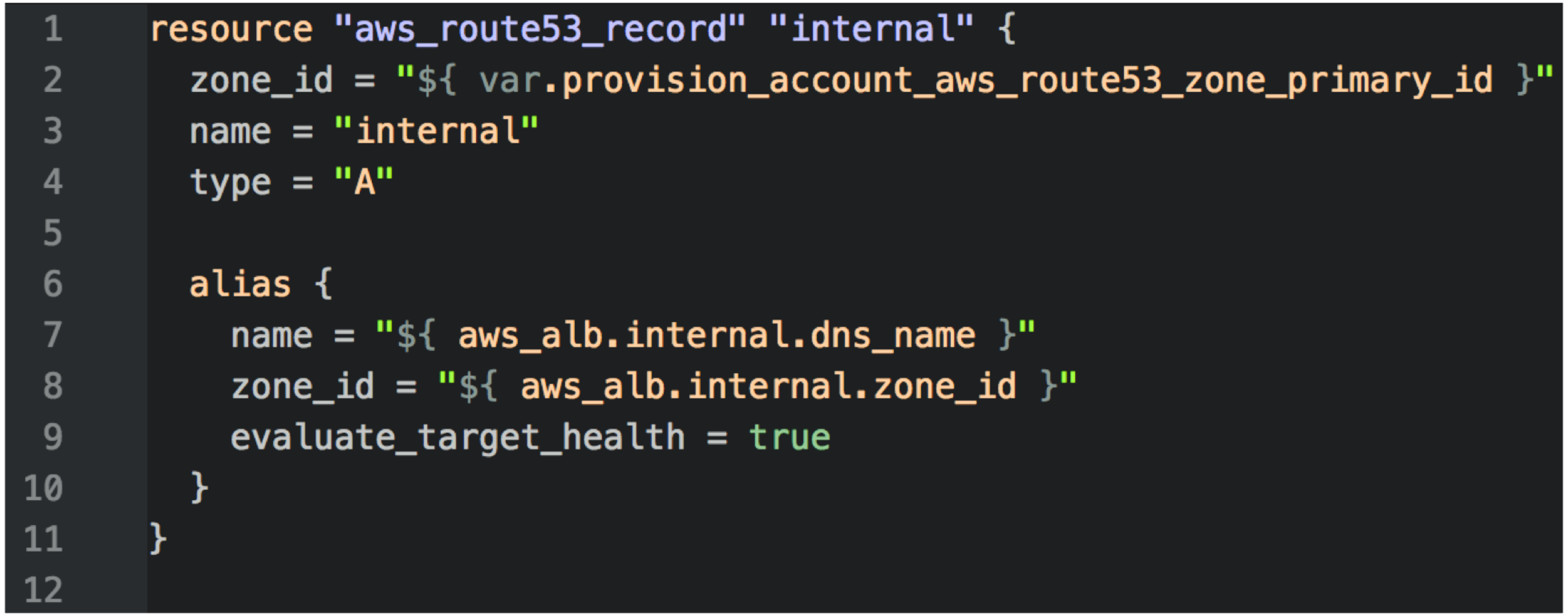
New Way

- Fully automated deploys via APIs
- Terraform for Infrastructure-as-Code
- “Share Nothing” environments
- Immutable builds
- Containerized deploy image

Infrastructure-as-code

Terraform

- Describe infrastructures in code
- Automatic calculation of diffs between deploys

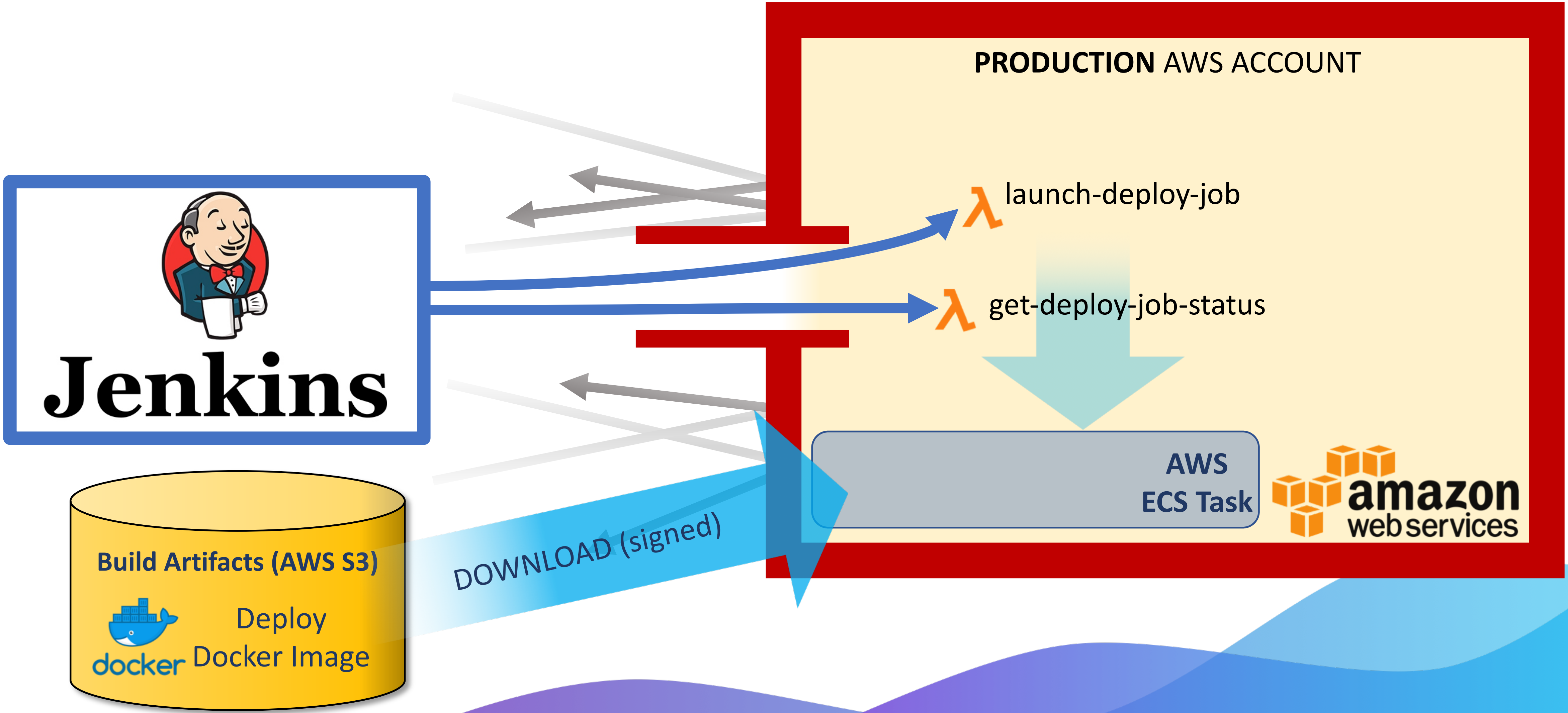


```
resource "aws_route53_record" "internal" {
  zone_id = "${ var.provision_account_aws_route53_zone_primary_id }"
  name = "internal"
  type = "A"

  alias {
    name = "${ aws_alb.internal.dns_name }"
    zone_id = "${ aws_alb.internal.zone_id }"
    evaluate_target_health = true
  }
}

```

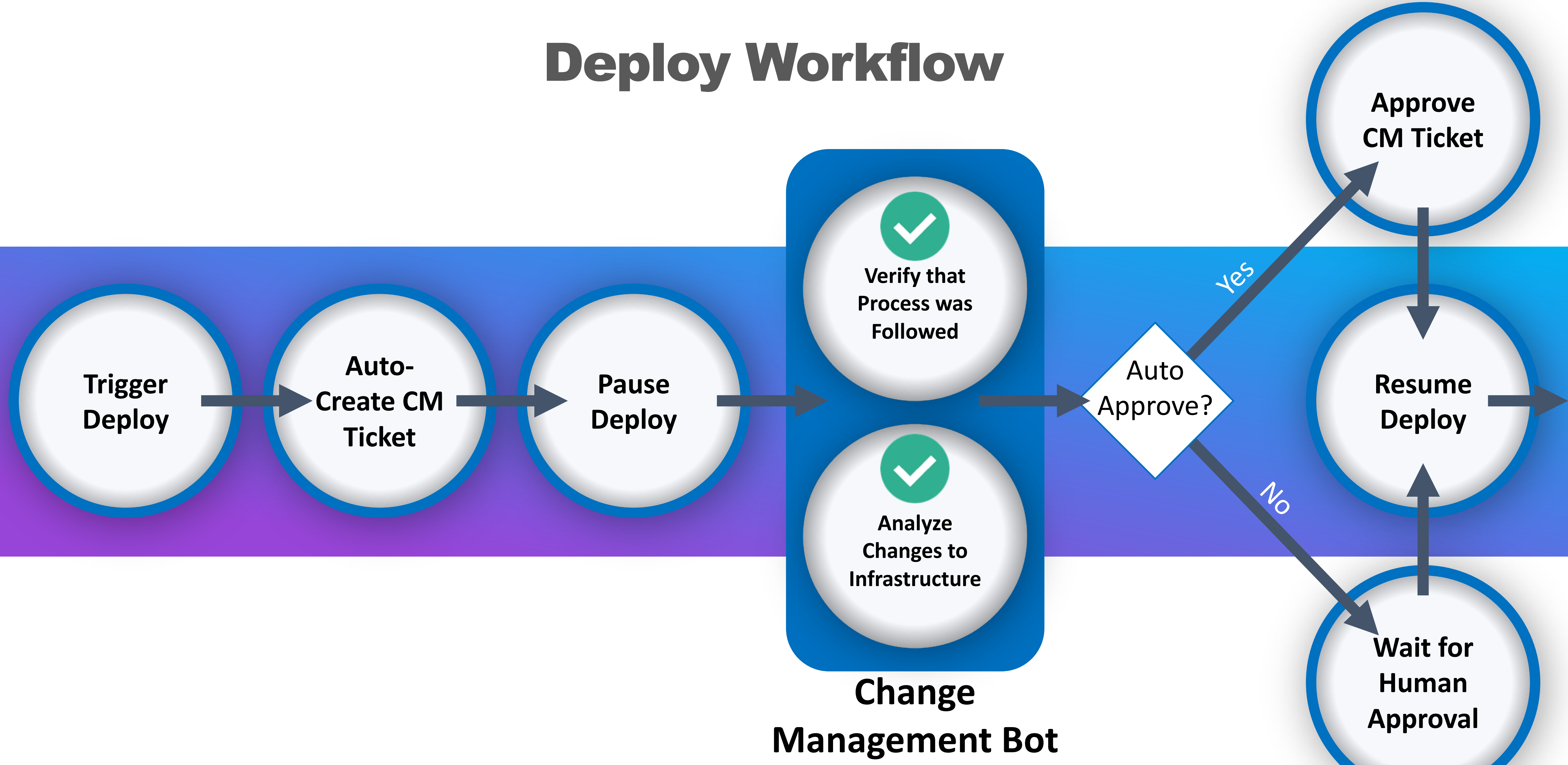
Environment Isolation



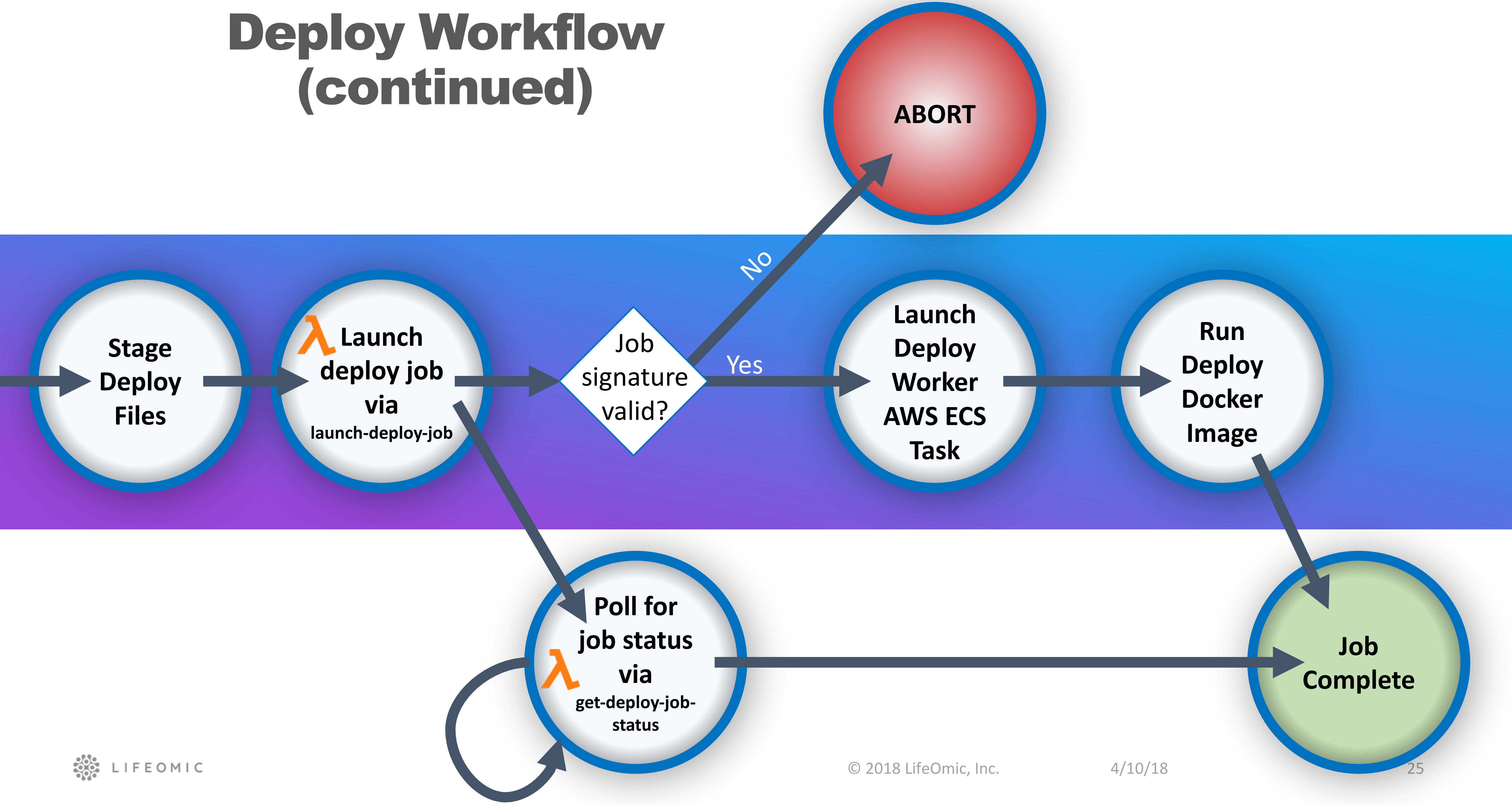
How do we ensure that this process has been followed with each production deploy?

What type of reviews and approvals are required and how does it scale with CI/CD in a Cloud DevOps operating environment?

Deploy Workflow



Deploy Workflow (continued)



Stage Deploy Files

Launch deploy job via launch-deploy-job

Job signature valid?

No

ABORT

Yes

Launch Deploy Worker AWS ECS Task

Run Deploy Docker Image

Poll for job status via get-deploy-job-status

Job Complete

4. The “bot”

Automating production change management review and approval (cm-bot)

Before the bot

Human Submitter

- How can I figure out what has changed since my last deployment?
- How much detail do I really need?
- Wait … wait … wait … ask somebody to approve

Human Reviewer

- Were the changes reviewed by others?
- Was a security scan run?
- Do I trust the list of changes? (Hint: You should not)

Life with a bot

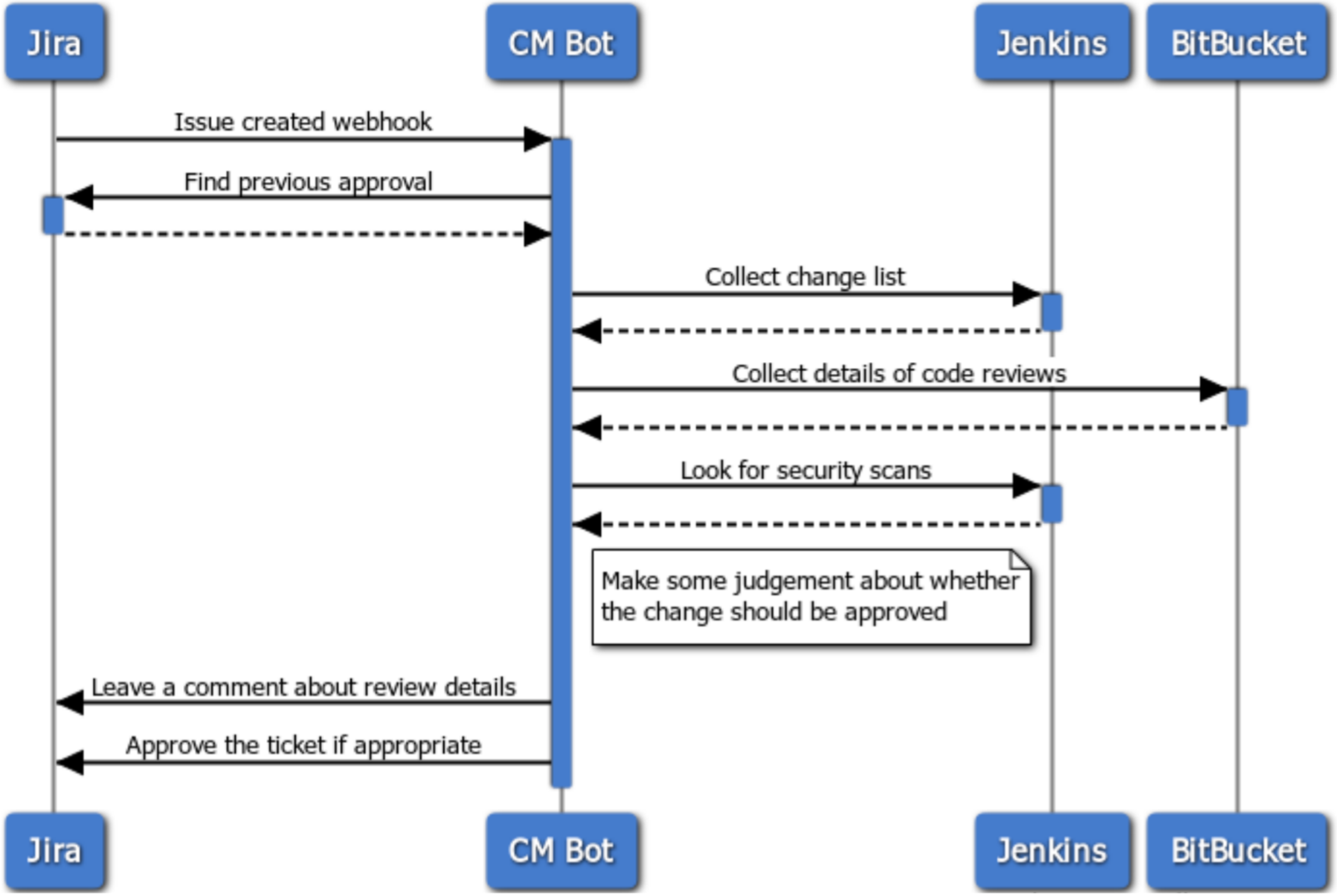
Human Submitter

- Provide summary text
- Provide Jenkins build reference

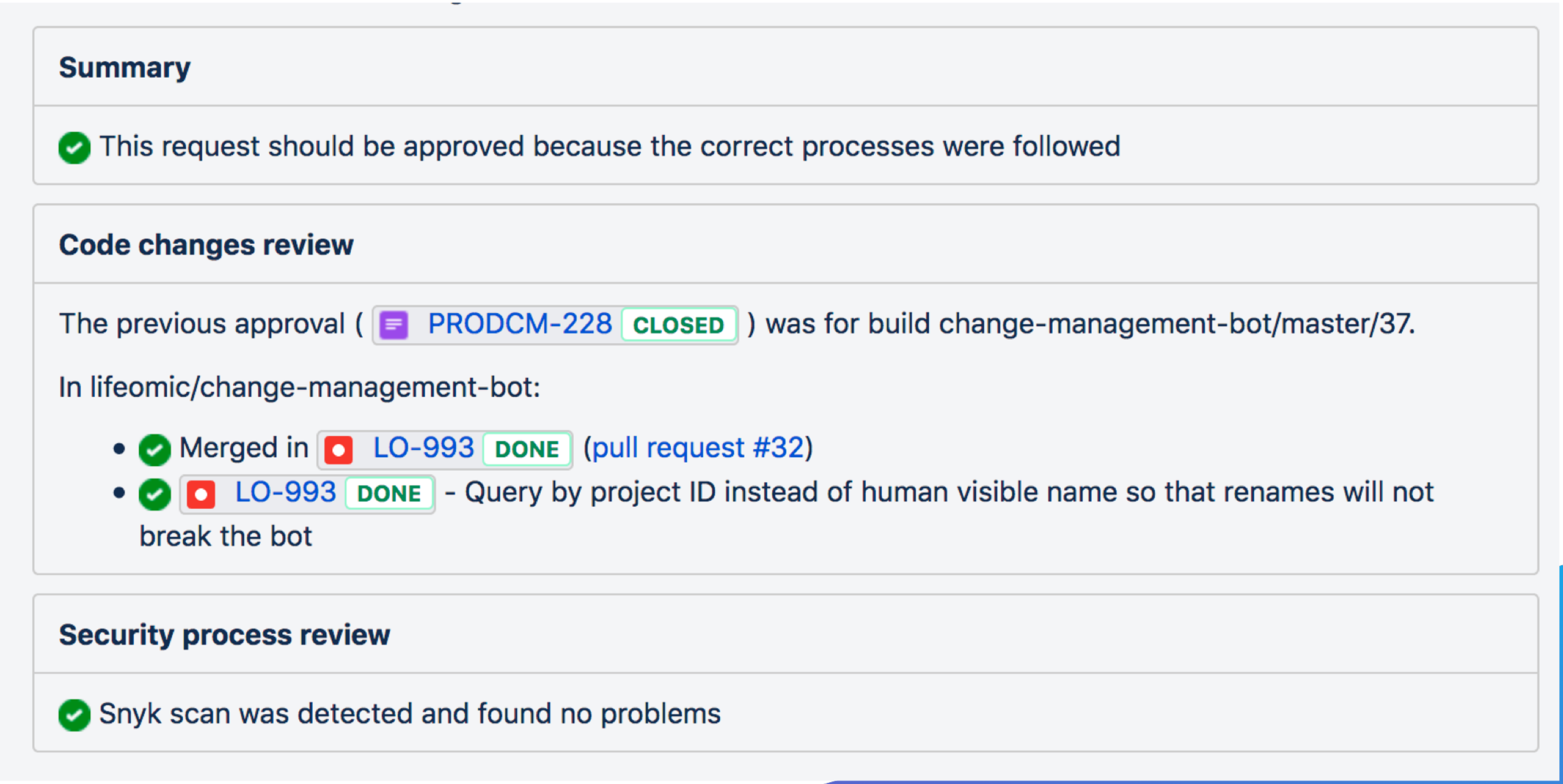
Automation

- Compute what changed since last deploy
- Verify changes were reviewed
- Detect security scanning
- Punt to human on problems

Automated approval workflow



Jira
CM Bot
Jenkins
BitBucket
Issue created webhook
Find previous approval
Collect change list
Collect details of code reviews
Look for security scans
Make some judgement about whether the change should be approved
Leave a comment about review details
Approve the ticket if appropriate
Jira
CM Bot
Jenkins
BitBucket



Summary

✓ This request should be approved because the correct processes were followed

Code changes review

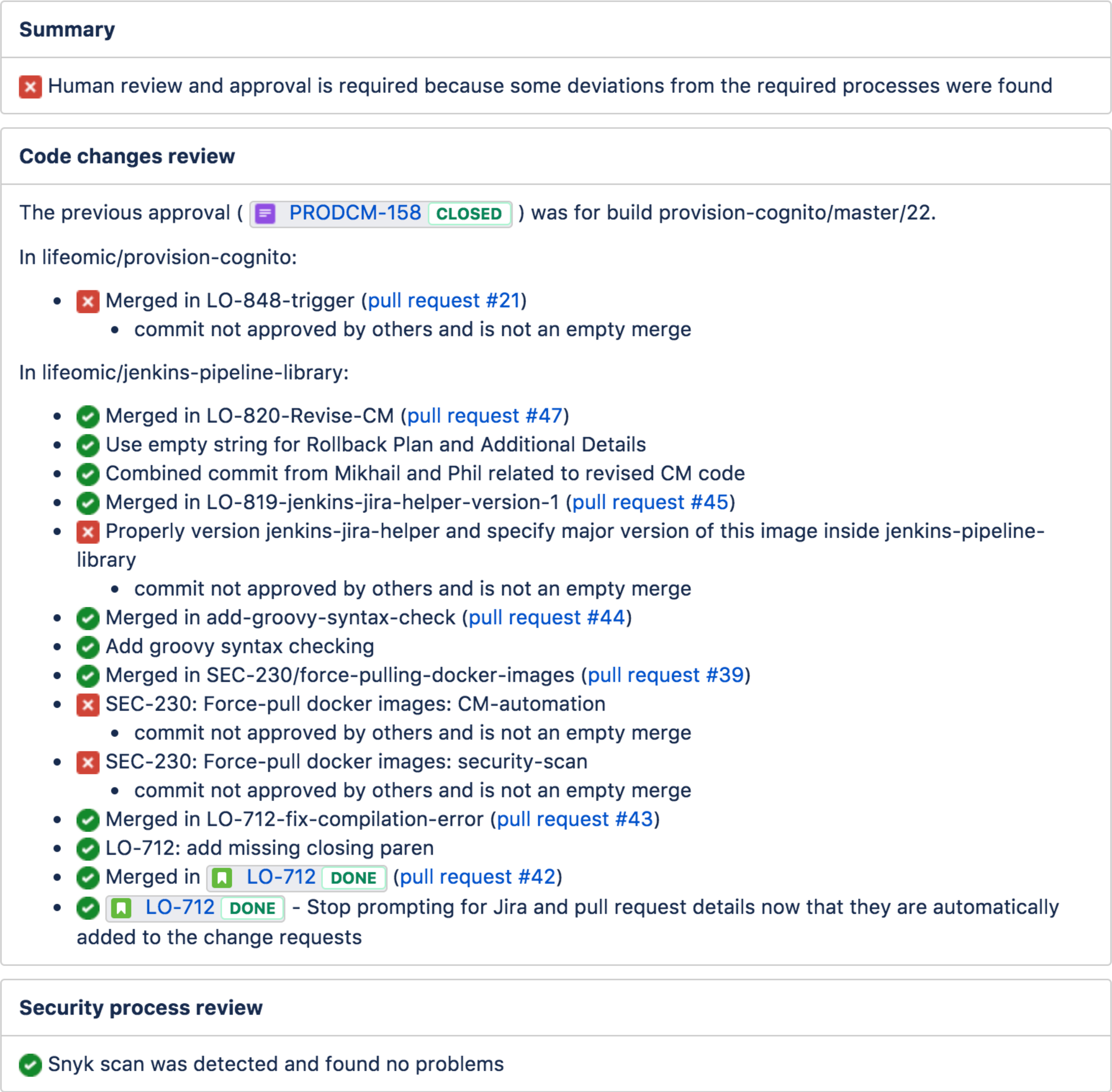
The previous approval (PRODCM-228 CLOSED) was for build change-management-bot/master/37.

In lifeomic/change-management-bot:

- ✓ Merged in LO-993 DONE (pull request #32)
- ✓ LO-993 DONE - Query by project ID instead of human visible name so that renames will not break the bot

Security process review

✓ Snyk scan was detected and found no problems



Summary

✗ Human review and approval is required because some deviations from the required processes were found

Code changes review

The previous approval (PRODCM-158 CLOSED) was for build provision-cognito/master/22.

In lifeomic/provision-cognito:

- ✗ Merged in LO-848-trigger (pull request #21)
  - commit not approved by others and is not an empty merge

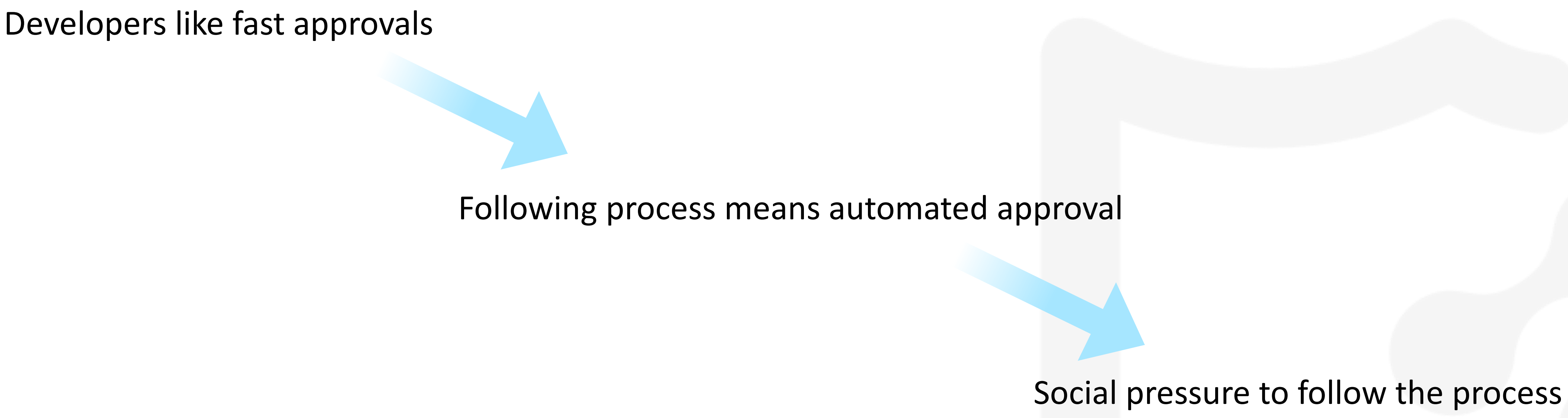
In lifeomic/jenkins-pipeline-library:

- ✓ Merged in LO-820-Revise-CM (pull request #47)
- ✓ Use empty string for Rollback Plan and Additional Details
- ✓ Combined commit from Mikhail and Phil related to revised CM code
- ✓ Merged in LO-819-jenkins-jira-helper-version-1 (pull request #45)
- ✗ Properly version jenkins-jira-helper and specify major version of this image inside jenkins-pipeline-library
  - commit not approved by others and is not an empty merge
- ✓ Merged in add-groovy-syntax-check (pull request #44)
- ✓ Add groovy syntax checking
- ✓ Merged in SEC-230/force-pulling-docker-images (pull request #39)
- ✗ SEC-230: Force-pull docker images: CM-automation
  - commit not approved by others and is not an empty merge
- ✗ SEC-230: Force-pull docker images: security-scan
  - commit not approved by others and is not an empty merge
- ✓ Merged in LO-712-fix-compilation-error (pull request #43)
- ✓ LO-712: add missing closing paren
- ✓ Merged in LO-712 DONE (pull request #42)
- ✓ LO-712 DONE - Stop prompting for Jira and pull request details now that they are automatically added to the change requests

Security process review

✓ Snyk scan was detected and found no problems

Incentives change culture



Developers like fast approvals

Following process means automated approval

Social pressure to follow the process

Summary and Next Steps

LESSONS LEARNED

- Existing DevOps solutions are unfortunately not “security-first”
- Our implementation has grown a bit too complex over time
- Influence positive culture change through automation and incentives
- Developers are not created equal – some code reviewers are more diligent than others but indistinguishable to the automation tools
- VPN access is overrated

FUTURE DEVELOPMENT

- Risky change detection in production deploys
- More intelligent rules or even ML to detect code changes in PRs (e.g. version bumps and package upgrades)
- Integrating SAST and DAST into the automation process
- Cross-platform, abstracted automation to help other organizations achieve the same security goals

LIFEOMIC

lifeomic.com