Nortal

DevDays

Designing Cloud-Ready Applications

Mike Ensor - May 2019

Introductions

- Started programming on Apple IIe
- 25 years in software
- Beta tested Google App Engine project (2008)
- Certified Google Cloud Architect (#170)
- AWS Certified Solution Architect
- Homebrewer
- Ice Hockey & Snowboarder (and I ski too)
+ Just relocated to Europe!

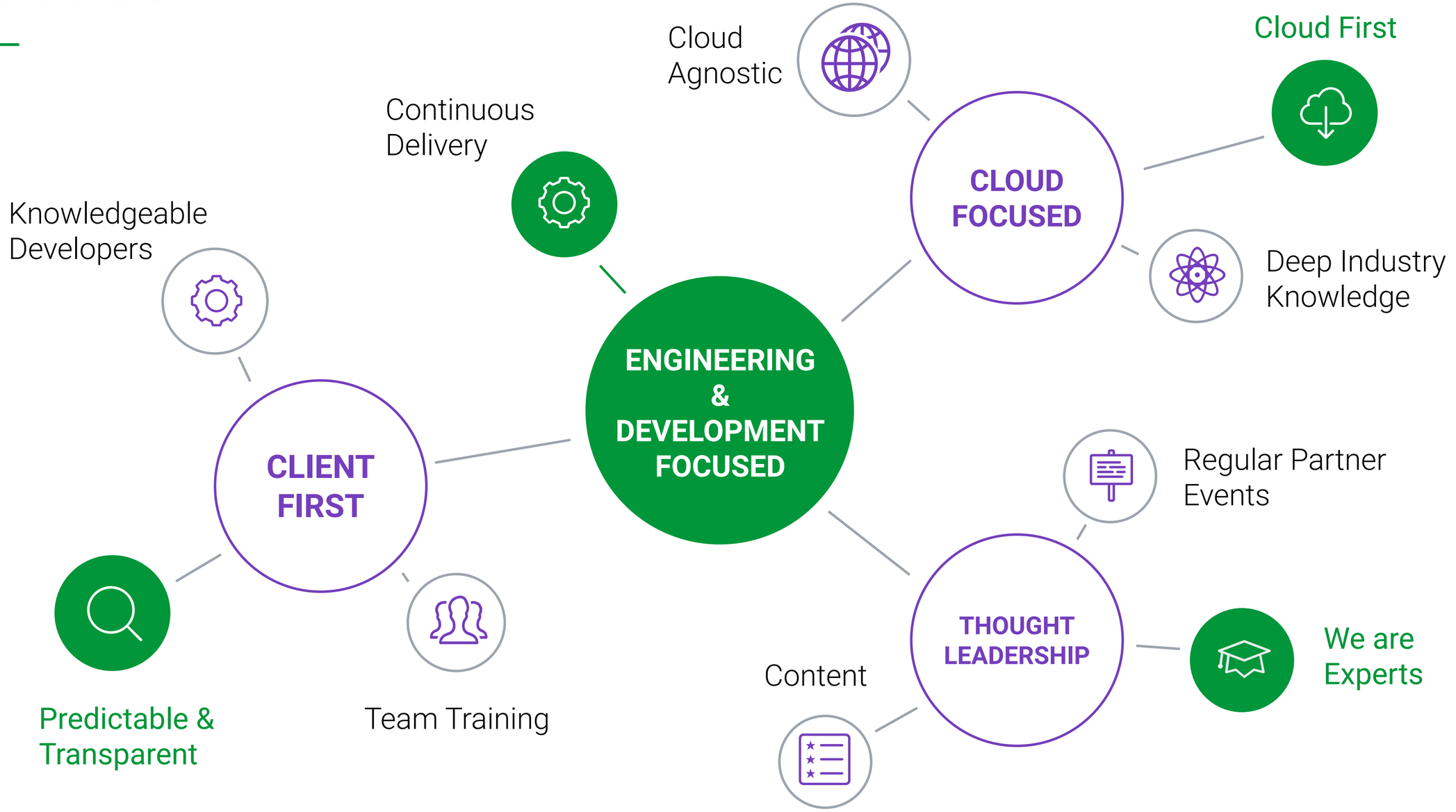
mike ensor

VP Global Cloud Practice

linkedin.com/in/mikeensor/

@mikeensor

About us



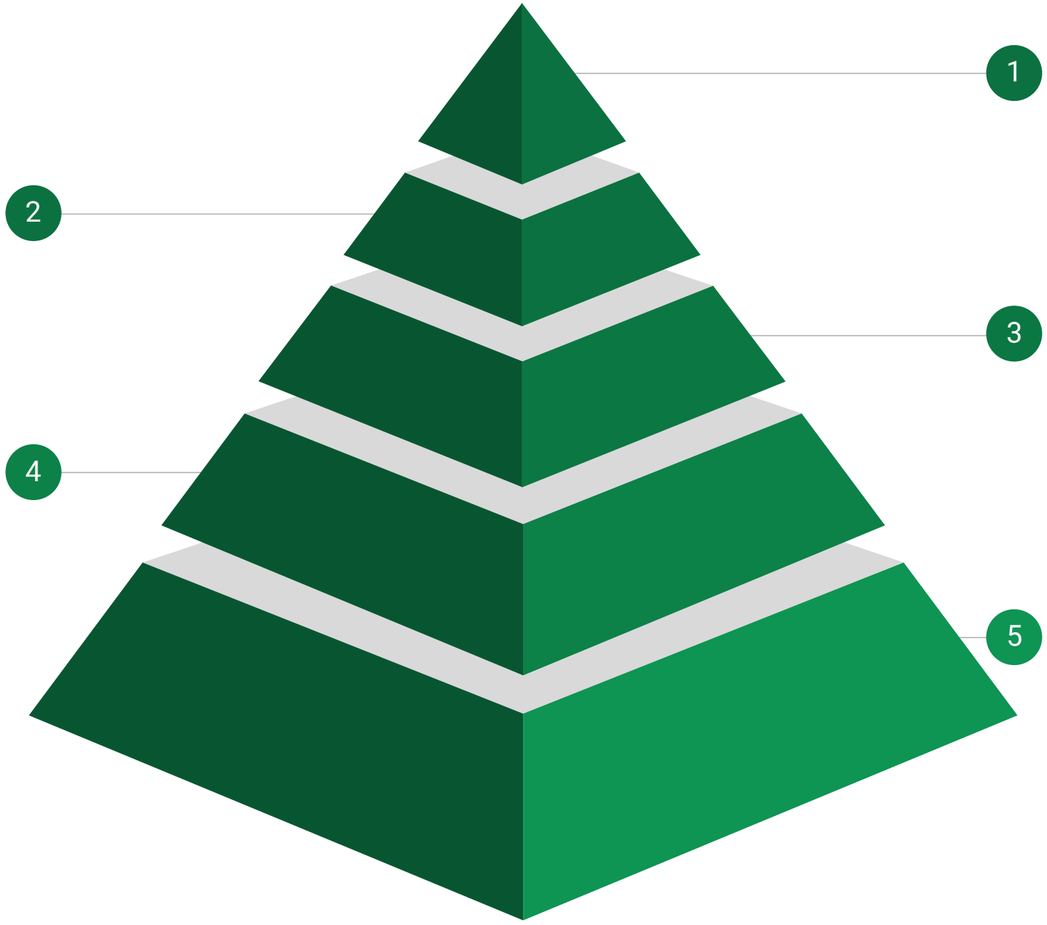
What we have planned for today!

1. What is “Cloud-ready”?
2. Differences to Traditional Delivery
3. Enhancing CICD w/ Security
4. Measuring maturity

“Cloud-native is an approach to building and running applications that exploits the advantages of the cloud computing delivery model.”

Nortal

Cloud Compute Layers



1

FaaS

Functions as a Service - Lambda

2

PaaS

Platform-as-a-Service removes infrastructure responsibilities allows developers to focus only on code

3

CaaS

Containers-as-a-Service reduces the constraints imposed in PaaS and focuses on orchestration of containers

4

Baked VMs

Immutable infrastructure at IaaS layer promoting better CI/CD practices

5

IaaS - Managed VMs

Infrastructure-as-a-Service provides most granular level of infrastructure services. Developers need to have greater knowledge of traditional IT operations

Cloud-Native

Apps Characteristics

1. **Visible**
   Ability to measure a service in your cloud
2. **Disposable**
   Services do not maintain state and can be removed at will
3. **Repeatable**
   Mature CI/CD pipeline produces predictable builds and deployments
4. **Follow 12-factor**
   Applications adhere to the 12-factor app
5. **Secure**
   Applications know how to utilize AuthN and AuthZ
6. **Distributed**
   New services can be found & accessed through a decoupled communication interface
7. **Performant**
   Applications are built to emphasize boot-up speed and focus on resource needs

Nortal

Traditional VS Cloud-Native

	Traditional	Cloud-Native
Operating System	OS Dependent	OS Abstraction
System Status	Un-predictable	Predictable
Capacity Planning	Oversized capacity	Right-Size / Elastic scale
Delivery Method	Waterfall	Continuous
Scaling	Manual	Automated (Intelligent)
Recovery	Slow recovery	Rapid / Auto-healing

Eight-Core Principles

1. Process must be repeatable & reliable
2. Automate everything!!
3. If something is painful, do it more often
4. Keep everything in source control
5. Done = Released!
6. Build quality in (make metrics visible!)
7. Everyone is responsible for the release
8. Improve continuously

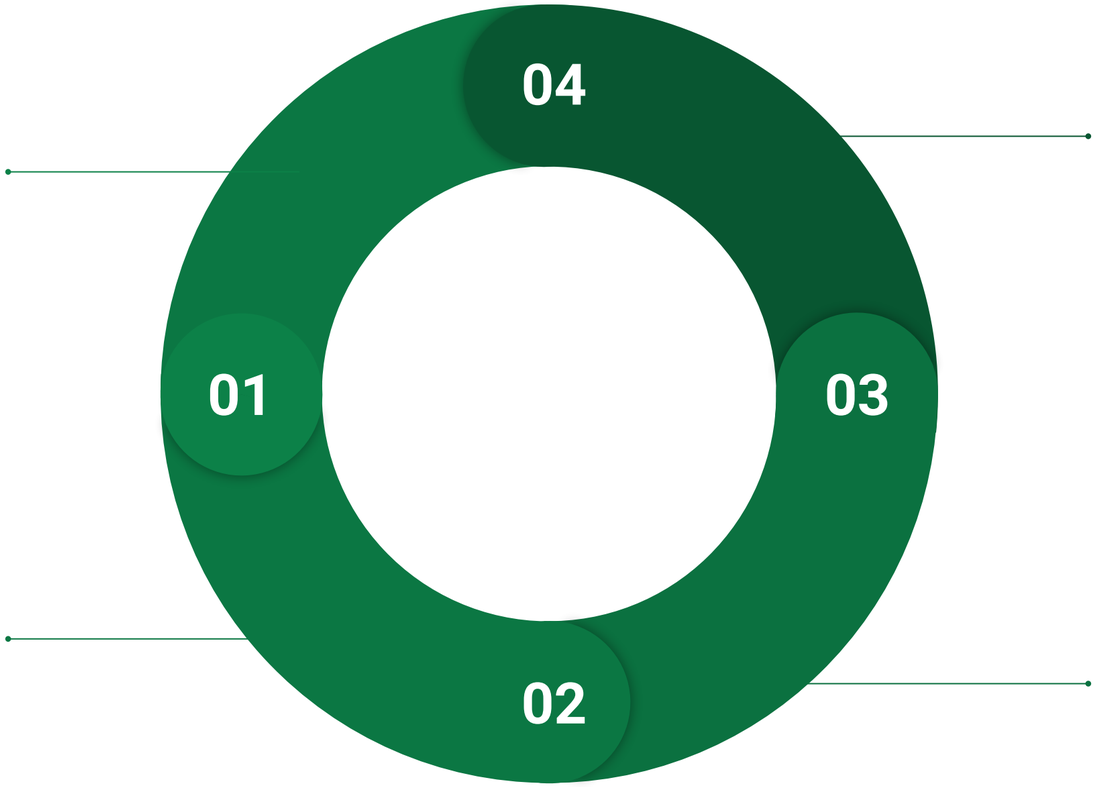
Four Core practices

build binaries once

Final binaries and/or packages should be versioned and immutable.

Use the same mechanism to deploy everywhere

Automate infrastructure for parity between environments; automate deployments to match each environment in progression.



Anything breaks, all hands on deck

Building software is the responsibility of the entire team. Any problems and everyone should help to solve the problem (and create tests to cover the cause)

Smoke Test / Deploy Frequently

Practice deployments often to avoid last minute or risky deployments to production.

Cloud Native using 12-Factor principles

Origins

- Designed by Heroku 2012
- Focus on stateless compute containers
- Best practices & guidelines, not rules
- Embodies Cloud Native best practices

Nortal

1 **Codebase**
One codebase tracked in revision control, many deploys

2 **Dependencies**
Explicitly declare and isolate dependencies

3 **Configuration**
Store config in the environment

4 **Backing Services**
Treat backing services as attached resources

5 **Build, release, run**
Strictly separate build and run stages.

6 **Processes**
Execute the app as one or more stateless processes.

7 **Port binding**
Export services via port binding

8 **Concurrency**
Scale out via the process model

9 **Disposability**
Maximize robustness with fast startup and graceful shutdown

10 **Dev/Prod Parity**
Keep development, staging and production as similar as possible

11 **Logs**
Treat logs as event streams

12 **Admin processes**
Run admin/management tasks as one-off processes.

Nortal

Cloud Native: Disposability

Pets

VS

Cattle

Changes for delivery

- Unique / Indispensable infrastructure
- Web/GUI driven "clicky"
- Tickets based
- Hand crafted
- Manual scaling
- Proprietary
- Snowflakes / Drift
- Vertical scale

- API driven
- Immutable infrastructure
- Self-Service
- Software-based Infrastructure
- On-Demand
- Heavy Automation
- Horizontal scale
- DevOps (GitOps) driven

Source Repository Strategy

GitHub Flow

Feature branch contains single changes; Use **Pull Requests** to merge back to master.

Master is production deployable.

GitLab Flow

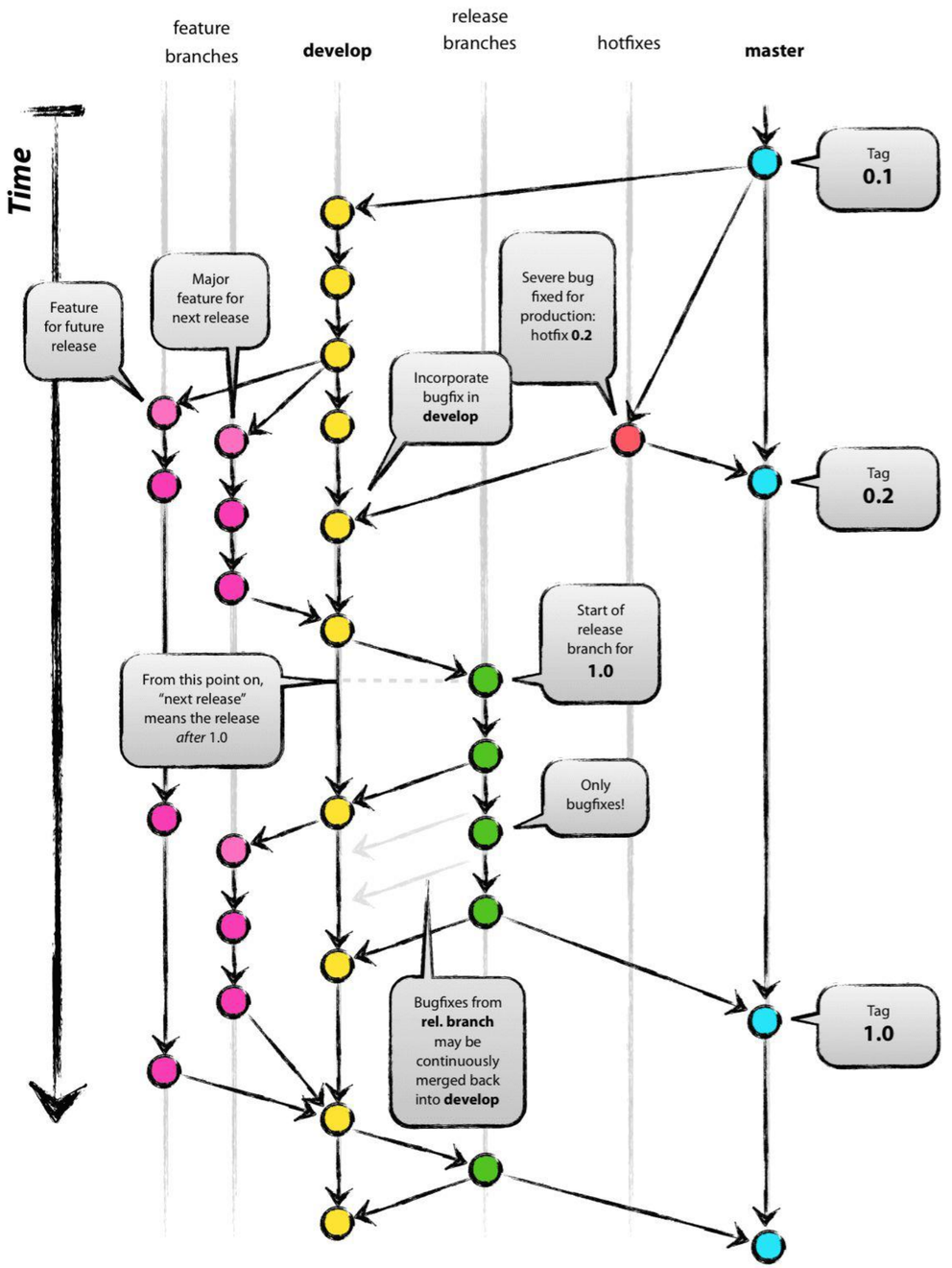
Feature branches merging back to master using **Pull Requests**. Master branch is active development **merging to “production” branch**.

Master is development, “production” branch is deployable.

GitFlow

Clone of SVN branching strategies. Tags, **long-running branches** containing **many features**. Branches deployed per environment. Special branch for production releases.

Git Flow Branching

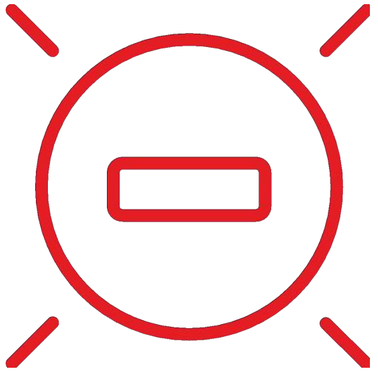


GitFlow: Pros VS Cons

Pros

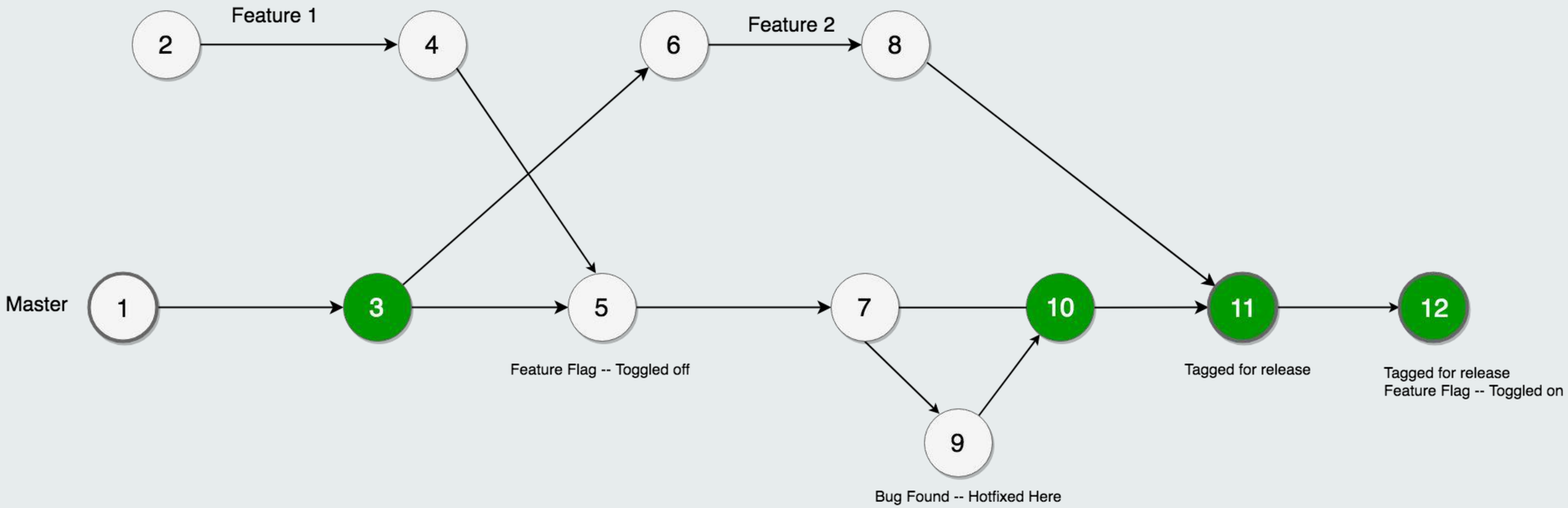
- Branches represent environments
- Familiar w/ SVN and older SCMs
- Ultimate feature isolation
- Low risk of releasing code early

Cons



- Very complex solution
- Pattern to recreate artifacts
- Requires mature teams
- Creates “merge” position
- Reintroduce bugs w/ bad merges
- Difficulty in recreating version

GitHub Flow Branching

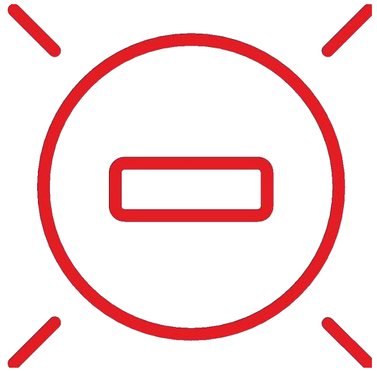


GitHub Flow: Pros VS Cons

Pros

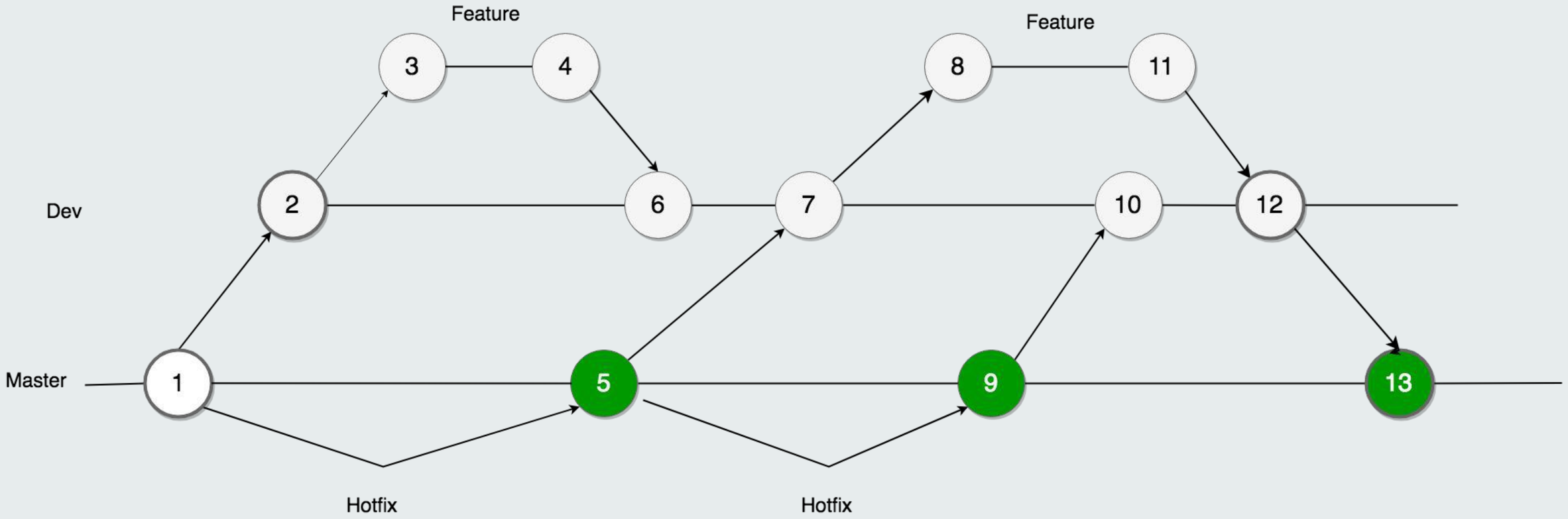
- Very simple
- Master is always production
- Encapsulates functional change
- Code reviews & codebase protection built into process
- Hotfixes applied immediately
- ONLY production artifacts

Cons



- Easy to prematurely release functionality
- Requires team discipline
- Responsibility on committer to address merge conflicts
- Advanced feature release needs coordination (ex: feature flags)

GitLab Flow Branching

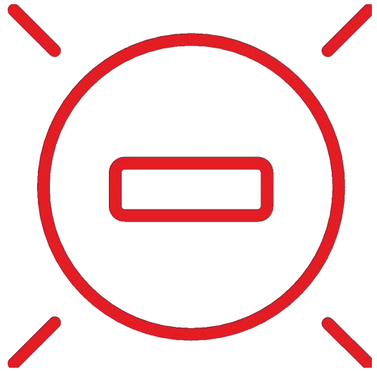


GitLab Flow: Pros VS Cons

Pros

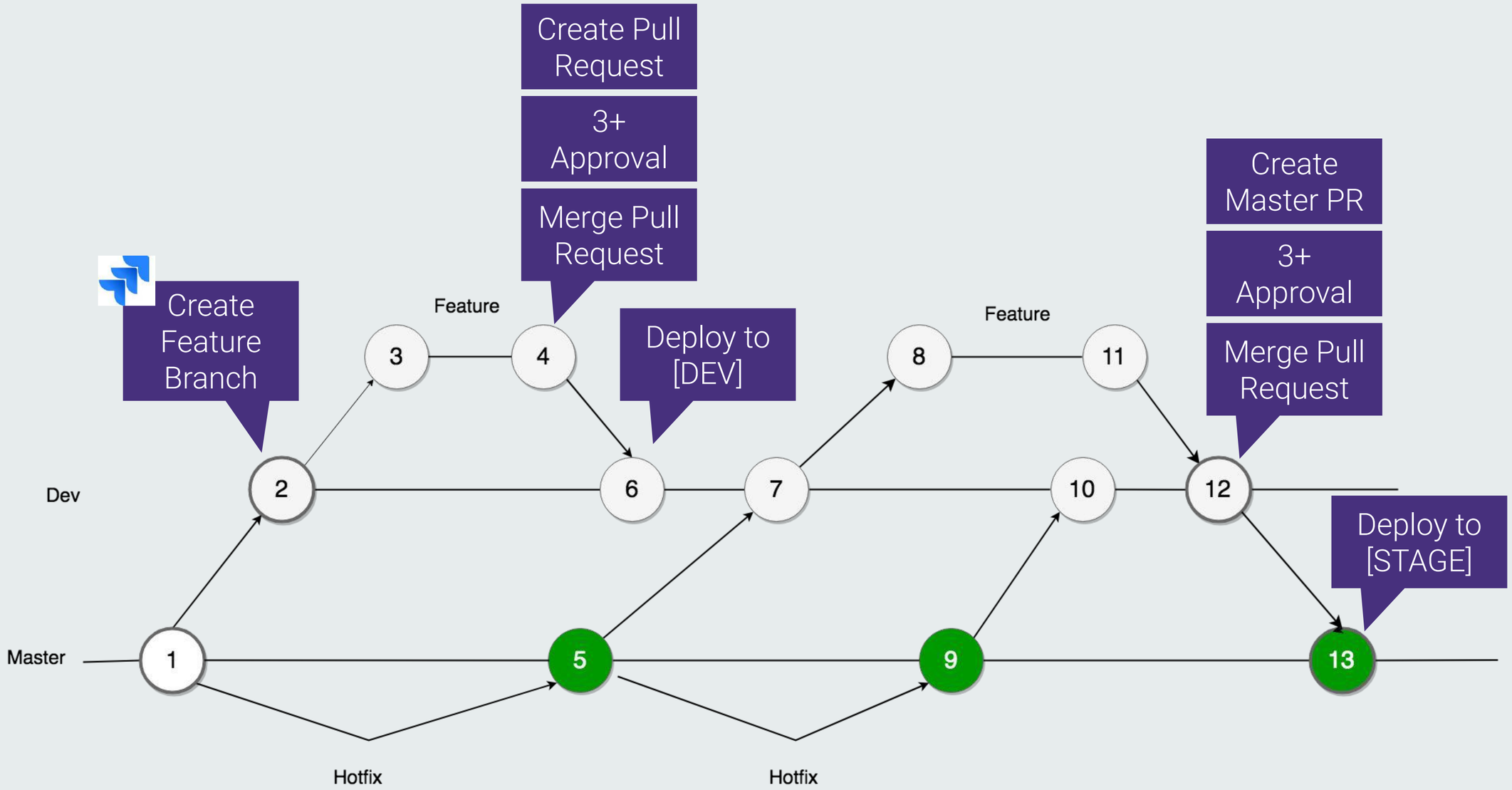
- Provides Sprint-level branches
- Master is always production
- Encapsulates functional change
- Code reviews & codebase protection built into process

Cons



- Creates non-production artifacts
- Requires team discipline
- Creates a need for branch hygiene

Adding & Building Code



Nortal

CI/CD Pipelines

Increasing Predictability & Reliability

Continuous Integration vs Continuous Delivery

Continuous Integration:

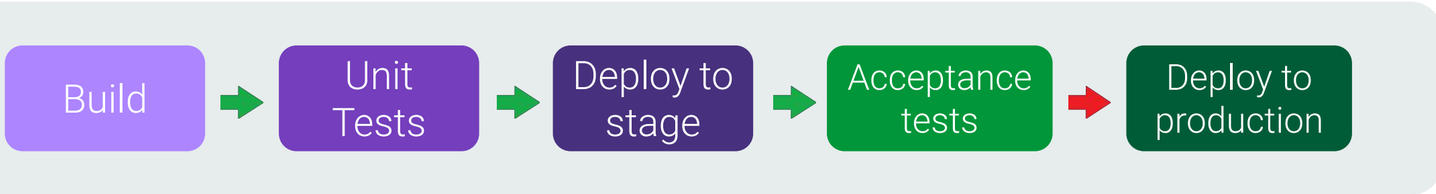
- **Tooling** used to integrate developers' code
- Compilation, Testing, orchestration
- Integrate all code contributions

Continuous Delivery:

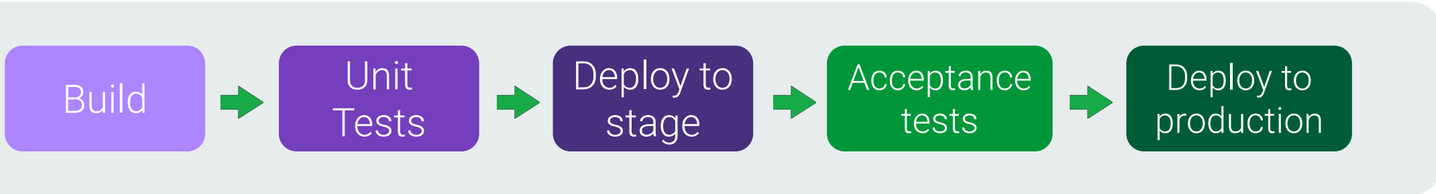
- Process, **not tooling**
- Not Continuous Deployment
- Logical stages implemented as "pipelines"

Continuous Integration

Continuous Delivery



Continuous Deployment



Automatic Manual

Tooling: Update CI/CD Pipelines

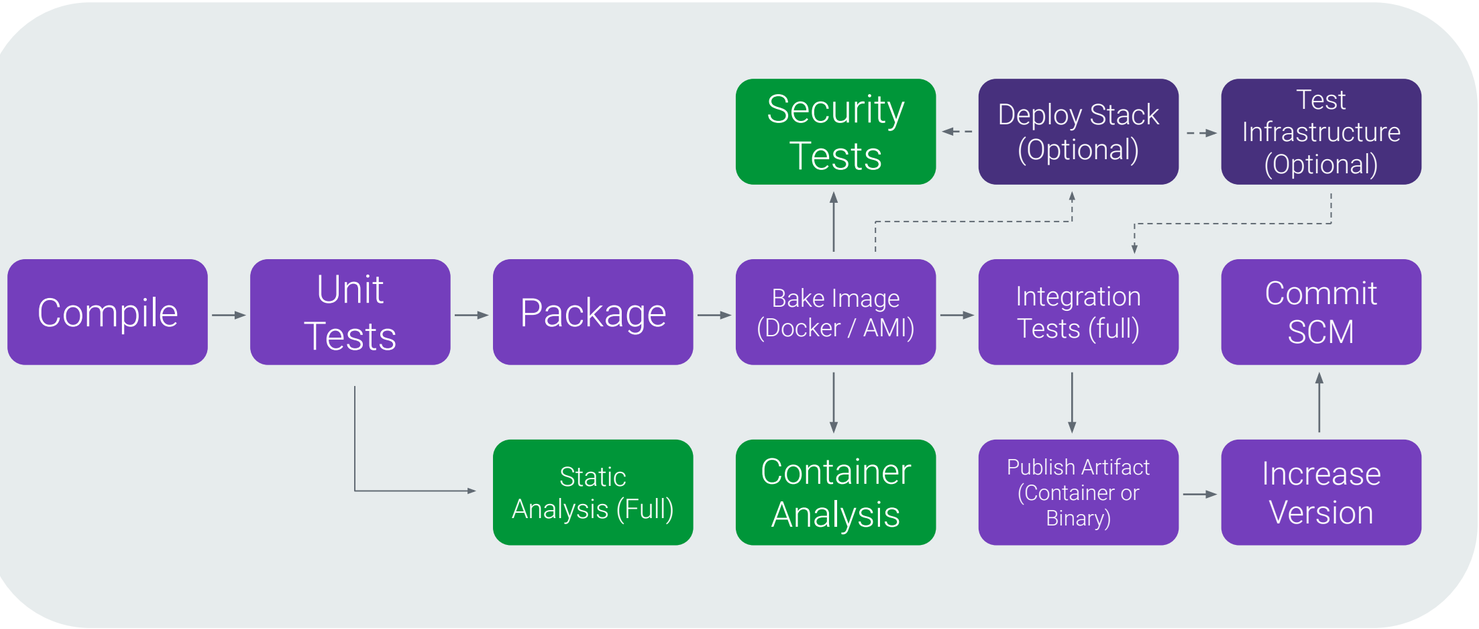
Continuous Delivery Pipeline:

- Code analysis
- Build / Compile
- Tests (unit, integration, etc)
- Release version
- Deployment
- Build Acceptance Tests (BAT)

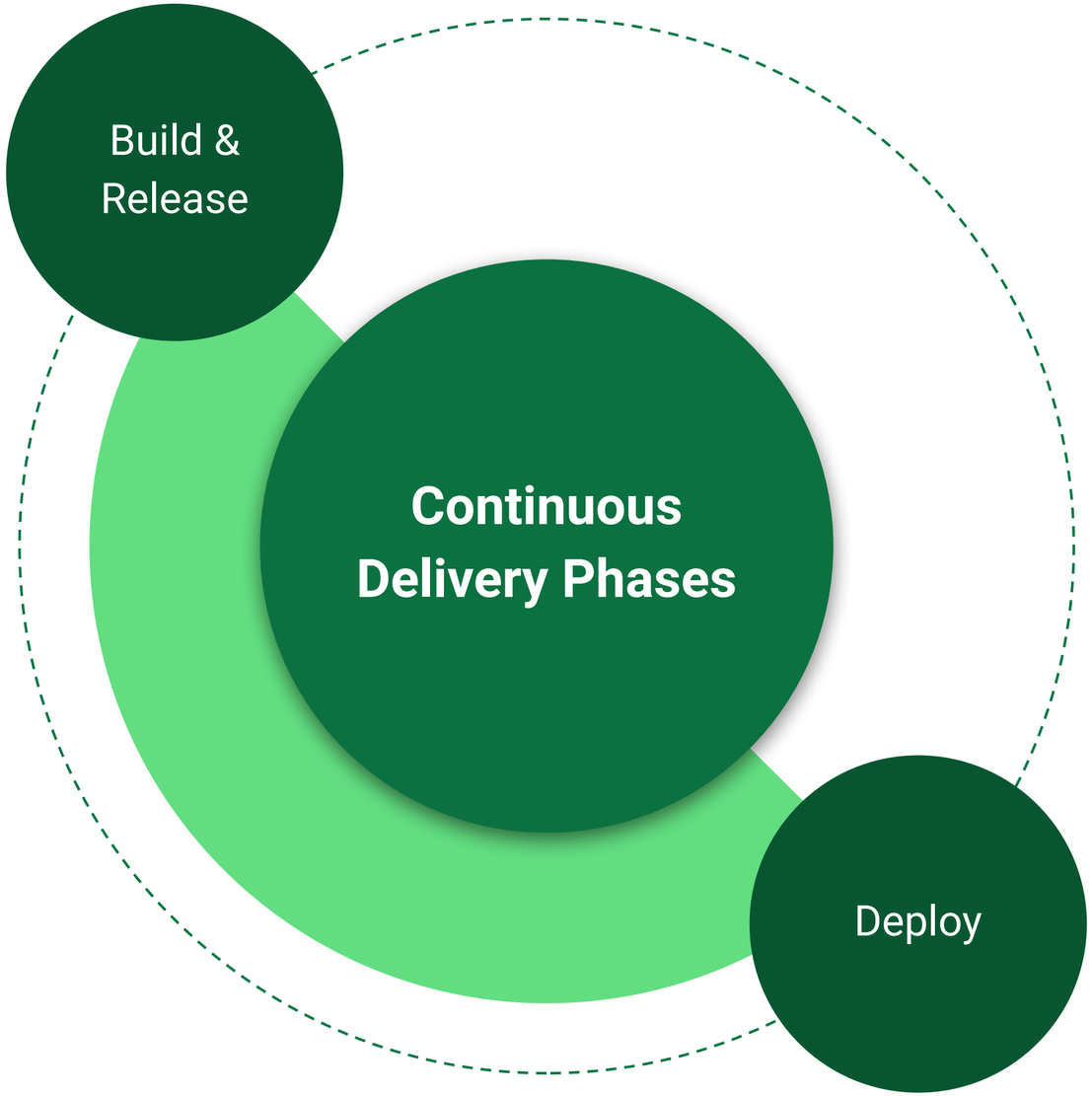
Modern Additions:

- Container & Image building
- Vulnerability scanning
- Parallelization
- Performance tests

Release Pipeline

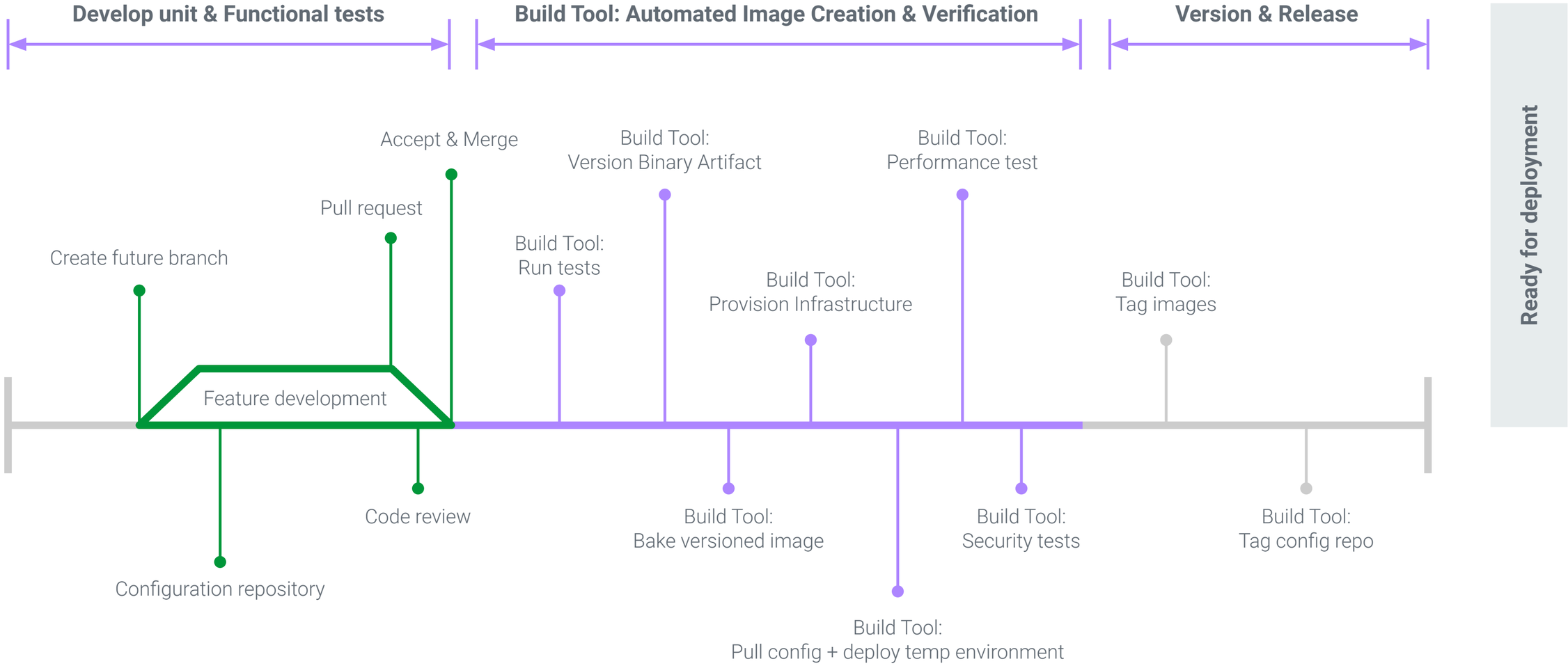


Continuous Delivery Phases

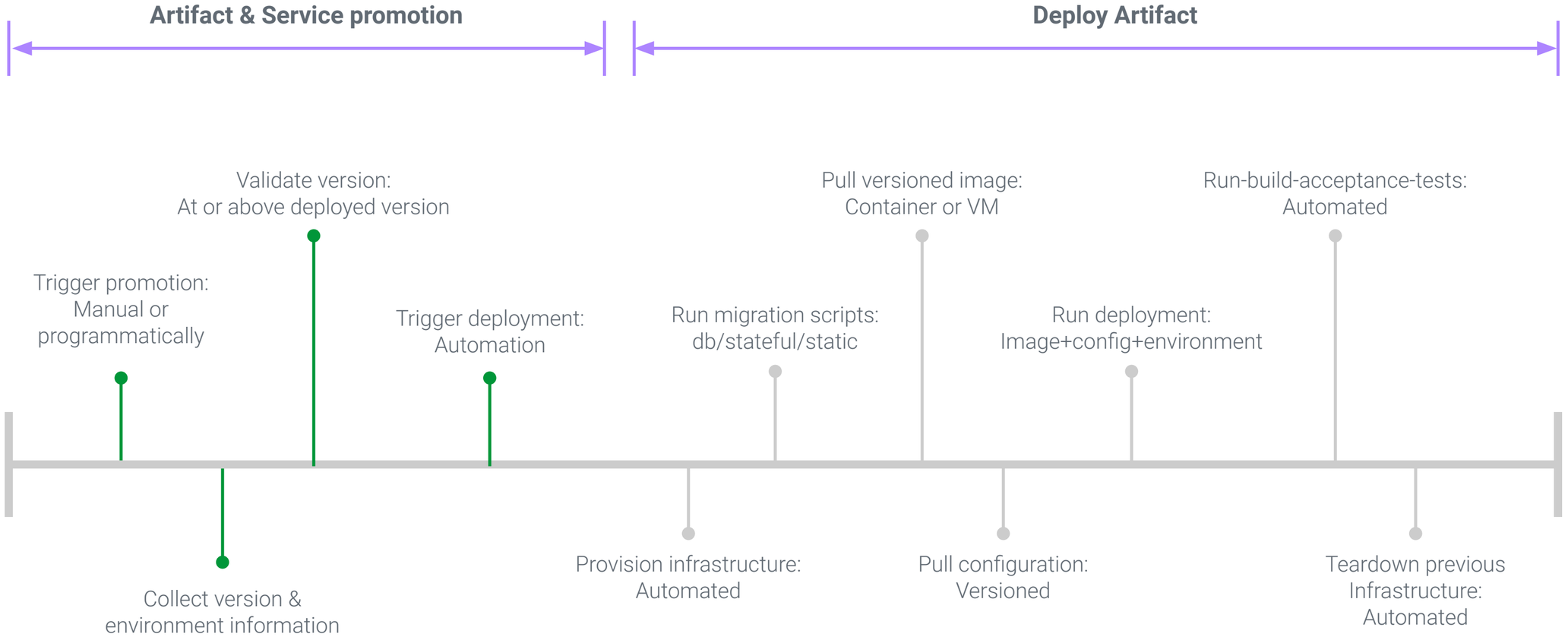


Nortal

Comprehensive CI/CD Pipelines: Release



Comprehensive CI/CD Pipelines: Deploy



Project Structure: CI/CD Pipeline

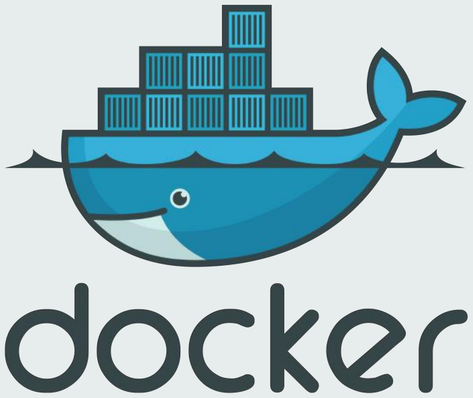
```
.
├── .git
├── .gitignore
├── Jenkinsfile
├── Dockerfile
├── Makefile
├── pom.xml
├── README.md
└── src
    ├── main
    │   └── java
    └── test
        └── java
```

Nortal

Folder Structure:

- CI/CD pipeline files
- Use **declarative format** files
  - Jenkins 2.x
  - GitLab CI
  - Bitbucket
  - CircleCI
- Promote consistency between services
  - Use **Builder Images** for Continuous Integration
- Keep pipeline files with repository
  - Builds are 100% built by pipeline
  - Delegate to tools from pipeline
- Dockerfile for image creation
- Create versioned docker image

Artifact Versioning



Version:

Short SHA hash

- Added to Image corresponding to git commit
  97dd2ae

Environment Marker (QA/Stage, Production)

- Added upon promotion out of environment
  dev-approved, qa-ready, qa-approved, etc

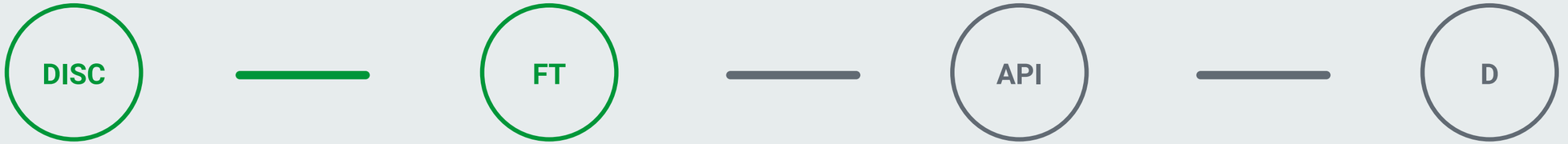
Version:

Semantic Versioning for helm-chart-service

- Increments when releasing Helm Chart
  1.0.1

Nortal

Cloud-Native: Distributed



Discoverable

Applications can easily be found once available

Fault Tolerant

Services handle errors and outage gracefully

API First Mentality

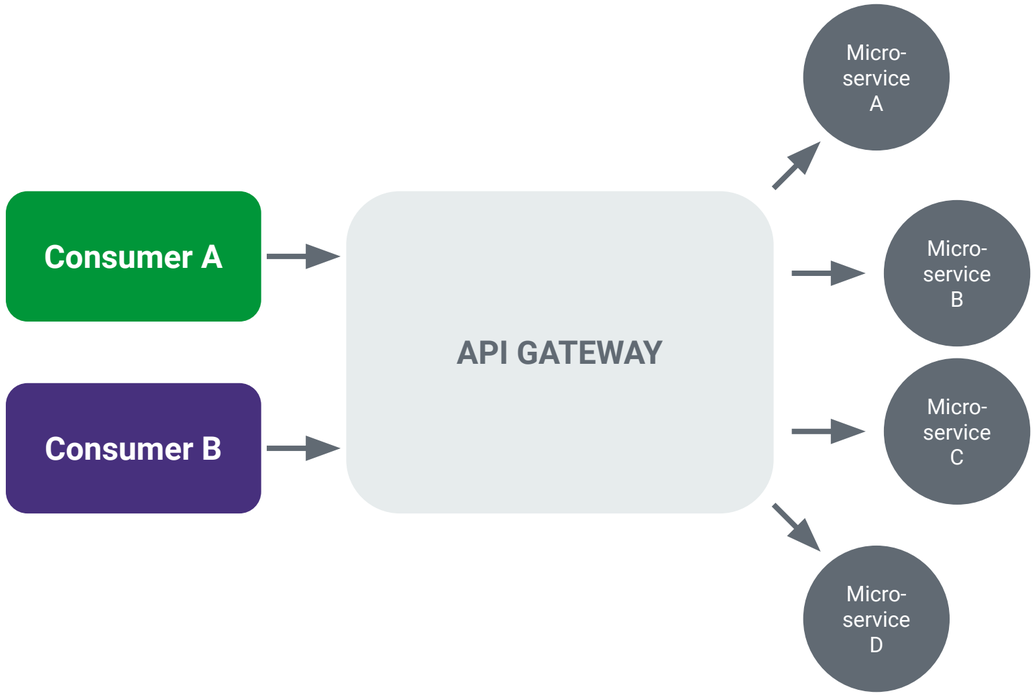
Design systems to be used via APIs

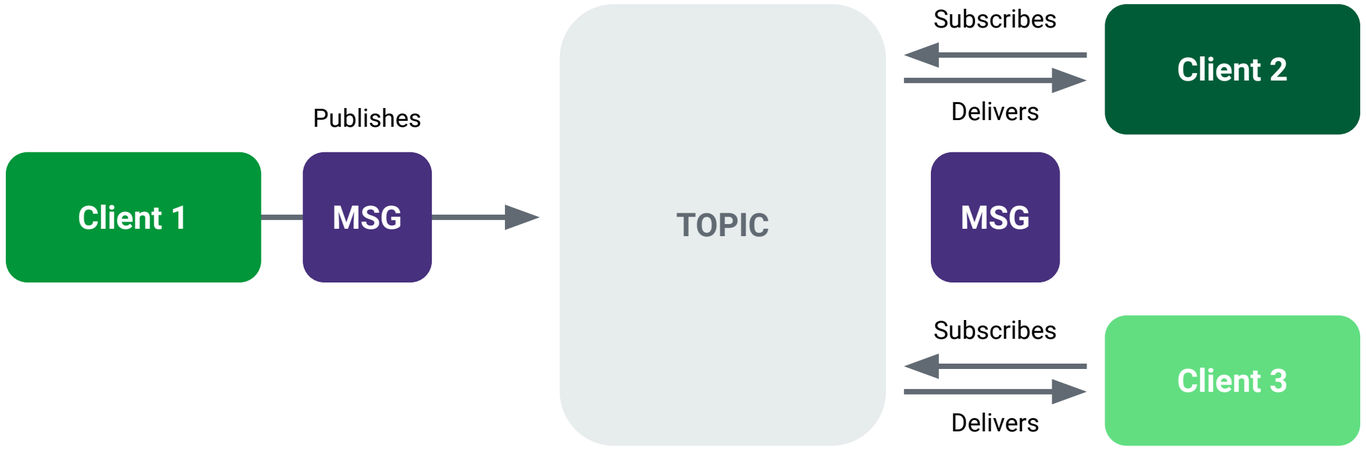
Decoupled

High Cohesion, Low Coupling using Queues and APIs

Distributed: API first & decoupling

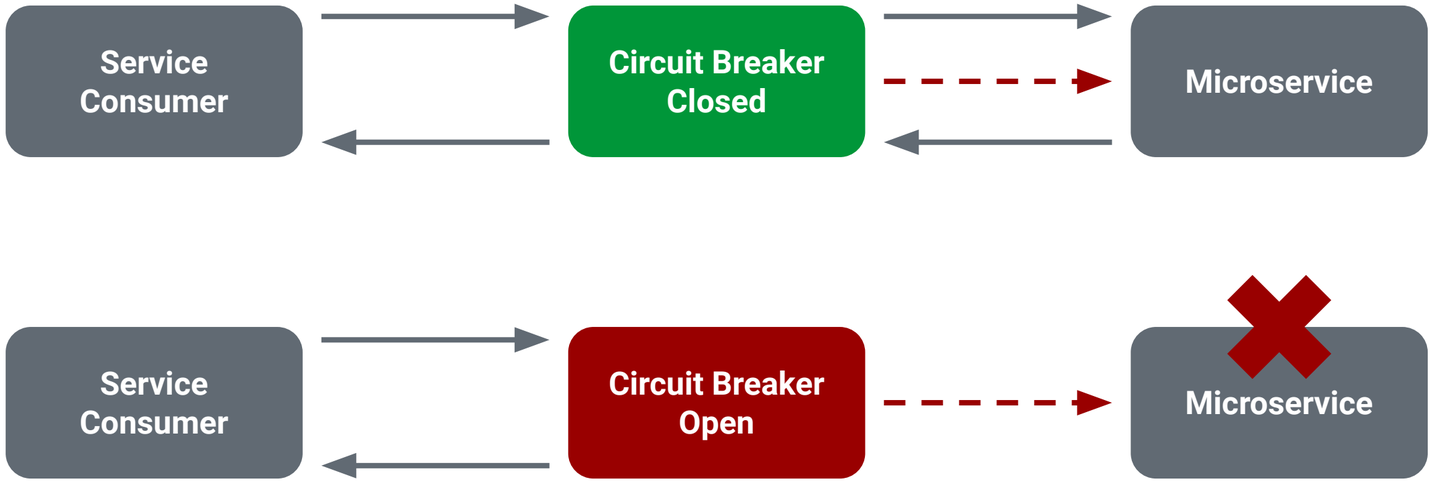
- Design for interfaces
  - Service contracts between services
  - Domain Driven Design
- Queues
  - **At-Most-Once** – Fire & Forget
  - **At-Least-Once** – Idempotency / Stateful
  - **Exactly-Once** – Stateful / Log streams
- Implement to APIs
  - TIP: If public facing, use and external API gateway on top of internal gateway
- API Gateways to decouple instances
  - Control access (authorization)
  - Govern performance & throughput





Distributed: Fault tolerant

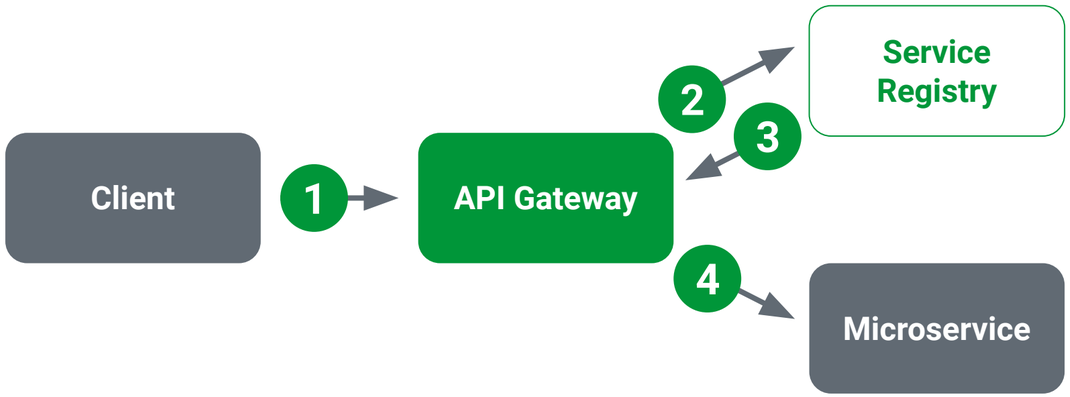
- Circuit Breakers
  - Programmatically define request thresholds
  - Inform upstream services of failure to serve
- Plan for failure
  - Automated testing at all levels
  - Implement Circuit Breakers & Bulkheads
  - Practice outages & disasters
- Implement **High Availability**
  - Multi-Zone & Multi-Region
- Use circuit breakers
  - Trip when conditions match
  - Disallow traffic to preserve overall service



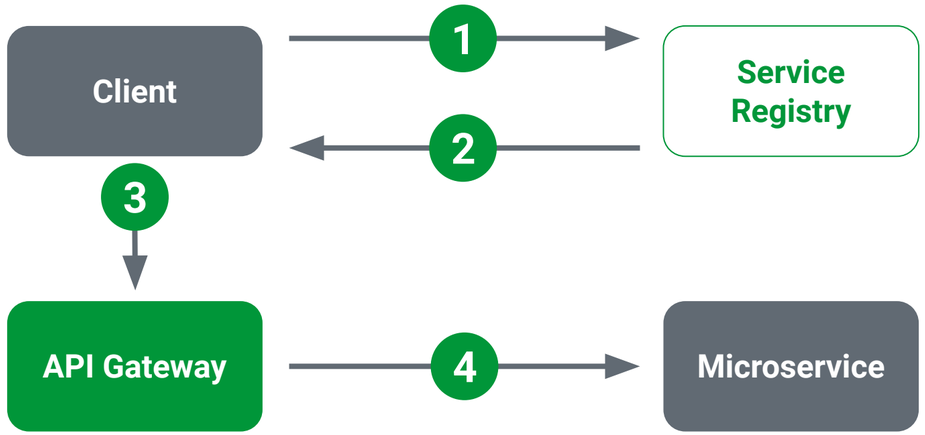
Distributed: Discoverability

- Rapid & Dynamic addition/removal of services
- Service Discovery
  - Catalog / Phonebook for system's apps
  - New services register
  - DNS vs Application-based
- Service Registries
  - Key/Value storage of services
  - Integrate with Load Balancers (if possible)
- Server-side VS Peer-to-Peer (Client-side) challenges
  - App: Single point of failure
  - Client: Consistency

Server-Side Discovery



Client-Side Discovery



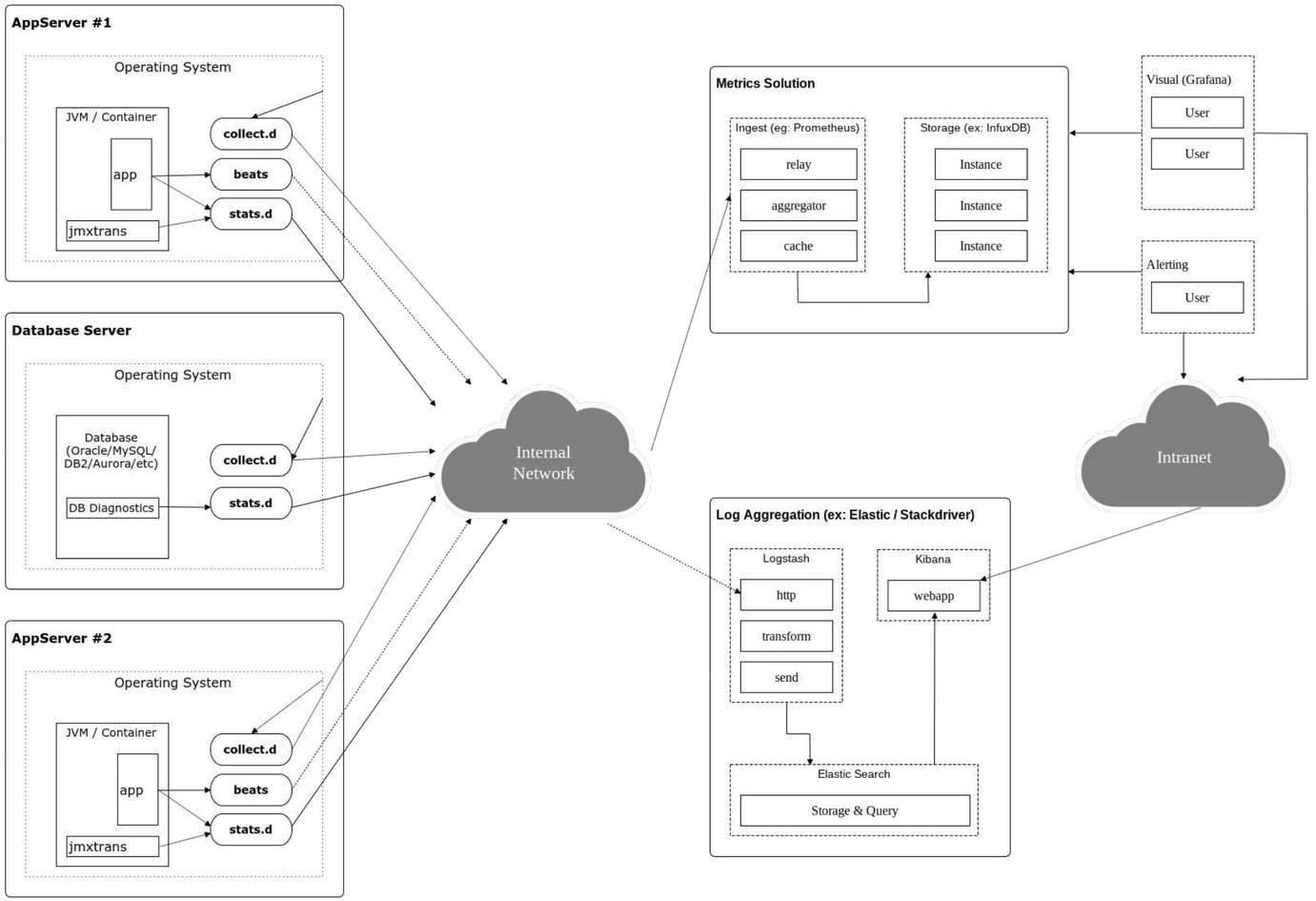
Observability & Visibility

Logging Considerations

- Centralized Aggregation & Authorization
- Embed context
- Establish log levels (trace/debug/info/warn)
- Engineer “Correlation IDs”
- **Tip:** Keep consistent log templates across enterprise

Metrics Considerations

- Business vs Systems Metrics
- Emit metrics over Logging metrics
- Support **S**ervice **L**evel **O**bjectives
- Alert on established thresholds (review often)



Use Synthetic metrics to validate system health

Cloud-Native Performant

TIP: Watch out for long call chains, could be incorrect Domain Design

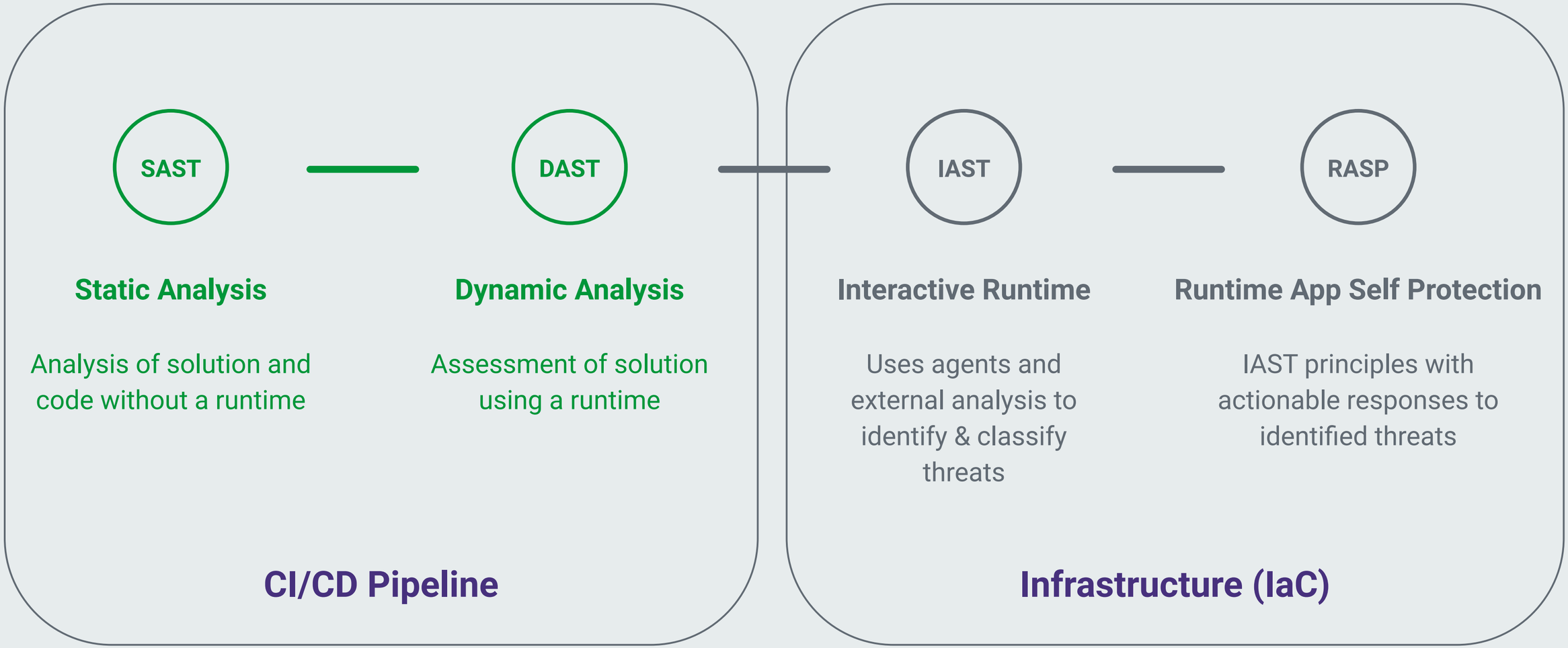
Nortal

Security Changes

Introduction of DevSecOps

Nortal

DevSecOps Categorization



Security: DevSecOps

DevSecOps:

- Separate automated testing suite
- Add to CI/CD Pipeline
- Layered approach using Automated Security Tooling
  - Static Code Analysis - SonarQube
  - Infrastructure - Gauntlet, BDD Security, Forseti
  - Application - ZAProxy, Veracode
  - Artifact - Maven Dependencies, Clair
- **SAST** - Compile & Artifact Creation
- **DAST** - Integration & BAT tests

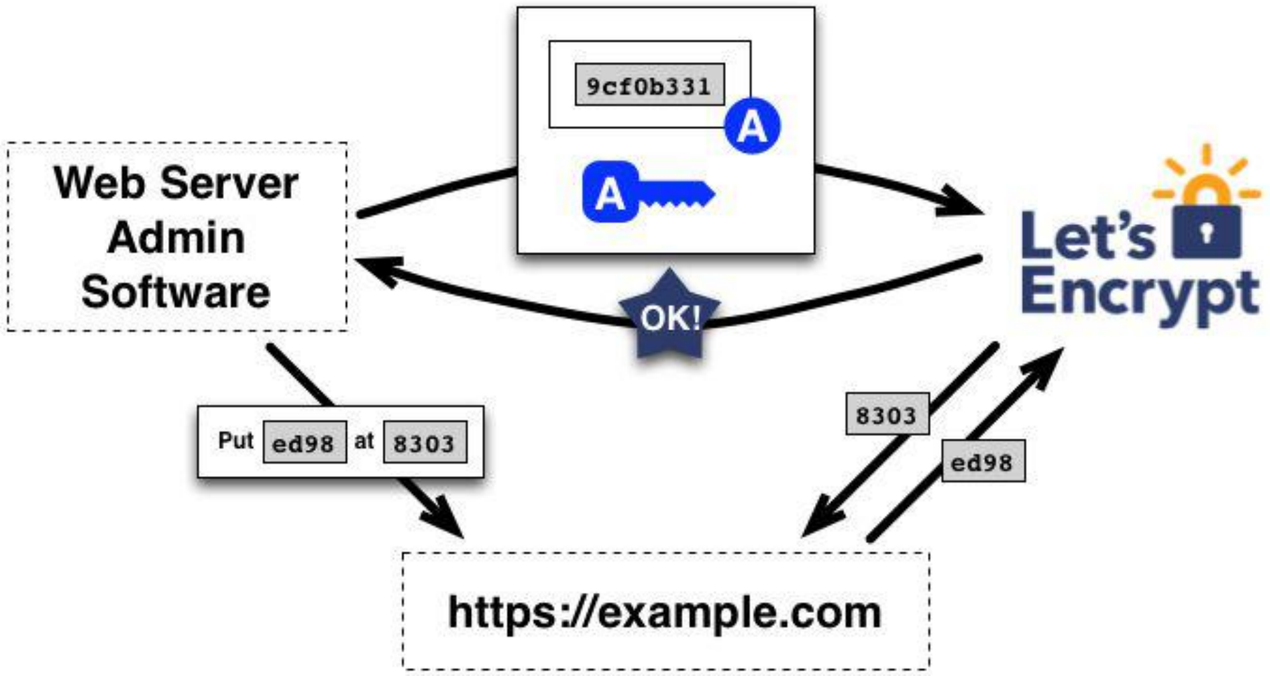
Example: Gauntlet Sample

```
# nmap-simple.attack
Feature: simple nmap attack to check for open ports

  Background:
    Given "nmap" is installed
    And the following profile:
      | name     | value        |
      | hostname | example. com |

  Scenario: Check standard web ports
    When I launch an "nmap" attack with:
      """ "
      nmap -F <hostname>
      " """
    Then the output should match / 80 . tcp \s + open /
    Then the output should not match:
      """ "
      25\/tcp\s+open
      " """
```

Cloud-Native Secure

Primary:

- Externalize Secrets
  - API-based, Deployment Integration
- External Configuration
  - ConfigMaps, Consul, etc
- External authentication Service
  - OAuth2 with JWT
  - Use scopes!
- Force HTTPS
  - ACME compliant
  - Certificate rotation & Revocation
  - Certbot & LetsEncrypt



Cert challenge for ACME protocol

Nortal

Cloud Native Maturity Matrix

Track application adoption

Cloud-Native

Maturity Matrix Categories

1

Design

Processes & tools associated with how a service designed

2

Build

How solutions are built and adhere to cloud-native principles

3

Test

Methodologies & Implementation tactics for ensuring quality & completeness

4

Deploy / Lifecycle

Defined capabilities controlling the deployment & lifecycle of services through automation

5

Monitor

Mechanisms & Tooling used enhance observability and adhere to SRE principles

Application Maturity Matrix

Team Evaluation:

- Workshop Procedure
  - Run with team regularly
  - Suggest every other sprint
  - Takes 15-20 minutes
- Make visible
  - Link results with repository
  - Track positive & negative progress
  - Influence backlog & tech debt
- Make it personal
  - Define YOUR maturity levels
  - Push your definition of “5”

Company Acme Inc
Application Widget Maker

Application Maturity Matrix

Category	Topic	01/02/2019	3/1/2019	4/8/2019	5/11/2019
Design	CI/CD Pipeline	3	4	5	5
	Feature Driven Development	4	4	4	5
	Just-in-Time Design	2	4	5	5
	API Contracts	1	4	4	4
	Defined Deprecation Cycle	1	1	1	2
Build	Trunk-based Development	5	5	5	5
	Feature Flags	3	4	4	4
	12-Factor Development	4	5	5	5
	Containers	5	5	5	5
	Configuration as Code	4	5	5	5
	Database Migration Strategy	4	5	5	5
Testing	Unit Test Coverage	5	5	5	5
	Automated Integration Tests	2	4	5	5
	Test Data Management	2	3	4	5
	Micro-benchmarking	1	1	2	3
	Static Source Scanning	5	5	5	5
	Dynamic Source Scanning	3	3	3	4
Deploy	Immuteable Artifacts	5	5	5	5
	Infrastructure as Code	4	5	5	5
	Push-on-Green (non-production)	5	5	5	5
	Production Deployment Method	3	3	3	4
	Deployment Method (B/G || Canary)	2	2	3	4
	Rollback Capabilities	1	1	1	1
	Automated Build Acceptance Tests	1	2	2	3
	Team Delivery Visibility	1	1	2	3
Monitoring	Distributed Tracing	2	2	3	3
	DevOps Telemetry	3	3	4	4
	Health Check Usage	5	5	5	5
	Performance Telemetry	4	4	5	5
	SLO Definitions	2	3	4	5
	Business Metrics Strategy & Impl	3	4	5	5

Nortal

Find out more (today & tomorrow)

Service Mesh	Matt Turner	Hall 1 - Thurs @12:15
Feature Flags	Andy Davies	Hall 6 - Wed @11:20
7 Stages Unit Tests	Jorge D. Ortiz Fuentes	Hall 1 - Wed @14:00
Chaos Engineering	Geert van der Cruijsen	Hall 6 - Thurs @14:50
Stranger Danger	Liran Tal	Hall 2 - Wed @12:15
Git as a database (just cool idea)	Kenneth Truyers	Hall 1 Thursday @11:20

Questions?

Presenter

mike.ensor@nortal.com
linkedin.com/in/mikeensor
twitter.com/mikeensor

Nortal online

nortal.com
linkedin.com/company/nortal
facebook.com/nortal
twitter.com/NortalGlobal