

A Brief Survey of the Web Deployment Landscape

Caleb Gosnell

Python Web Conference 2020



Expectations

This talk might help if you are:

- new
- overwhelmed or confused
- feeling the fear of missing out

Agenda

- ➔ Preliminaries
 - A General Model
 - Catalog of Practices
 - Example Workflows
 - Considerations, Caveats, and Advice



Caleb Gosnell

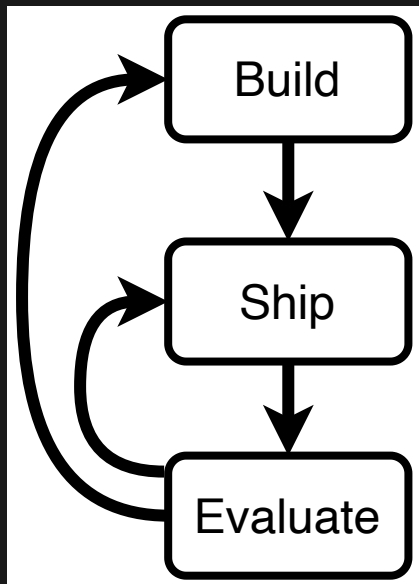
DevOps Engineer at
Six Feet Up

Trying hard to manage
computers and networks,
and deploy other people's
code since 2008.

Agenda

- ✓ Preliminaries
- ➔ A General Model
 - Catalog of Practices
 - Example Workflows
 - Considerations, Caveats, and Advice

Model Deployment Steps

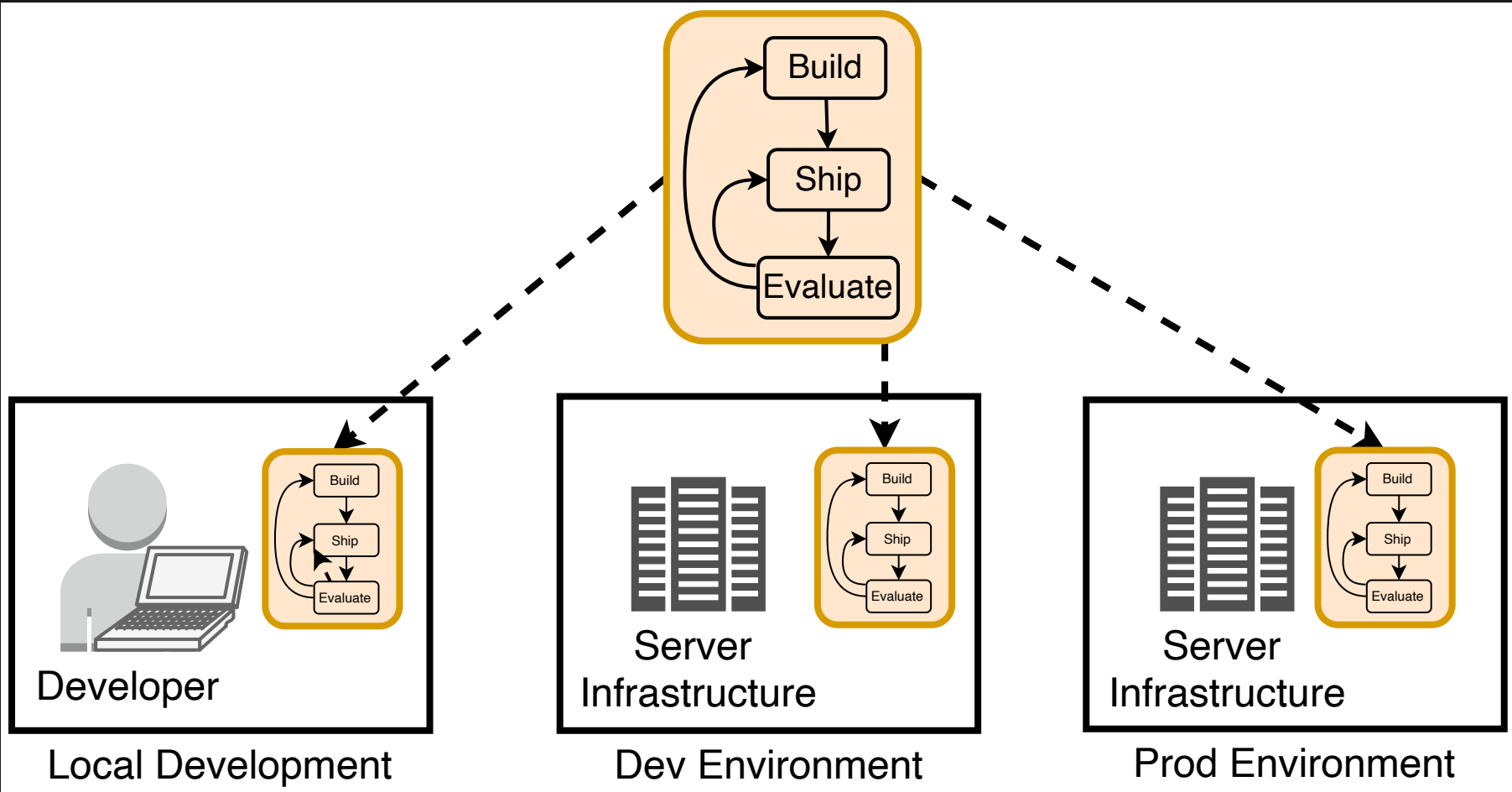


Build: produces a release/artifact

Ship: launches a release

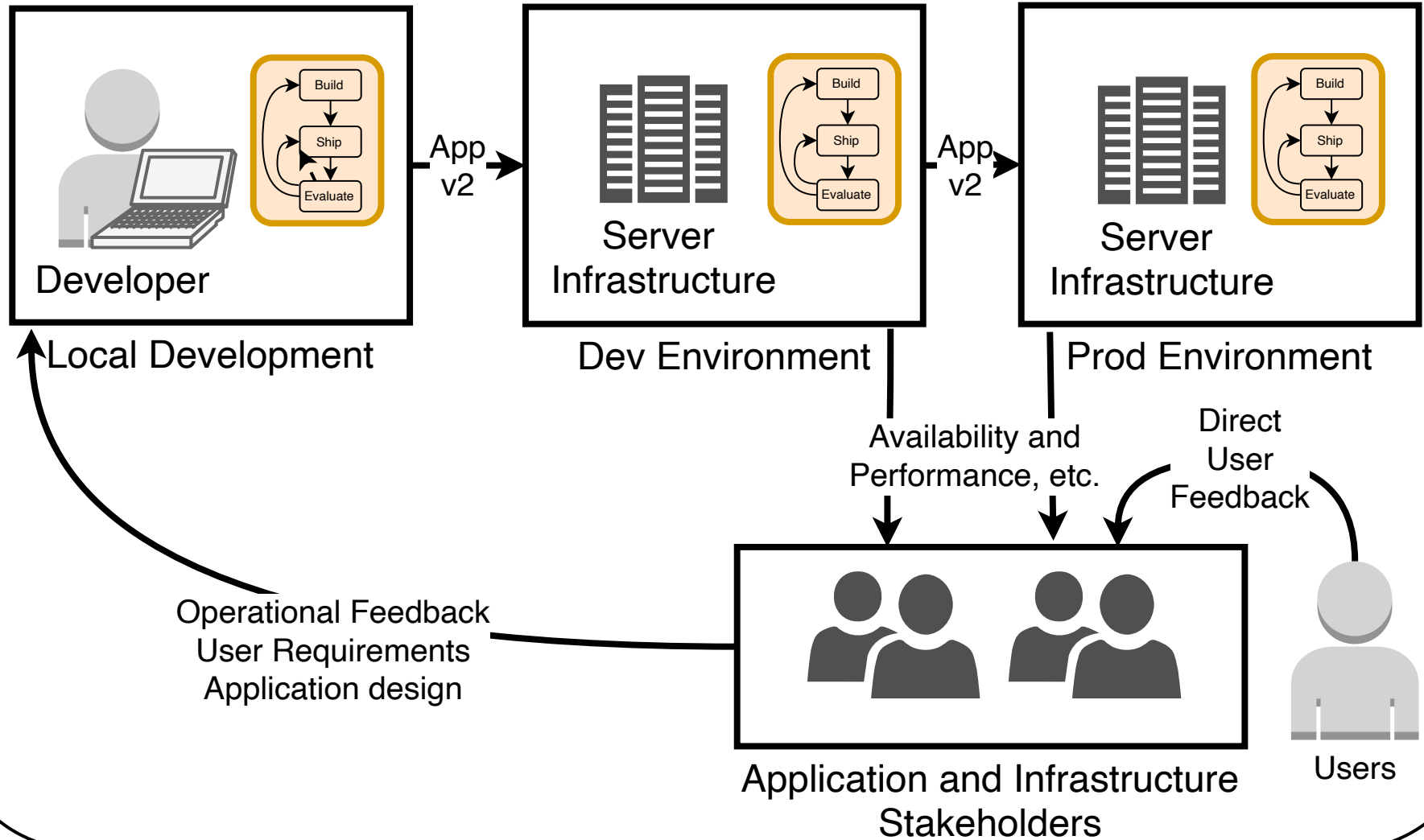
Evaluate: collects information
resulting from the release

Everything is a Deployment



Deployment Context - SLDC

Software Development Lifecycle

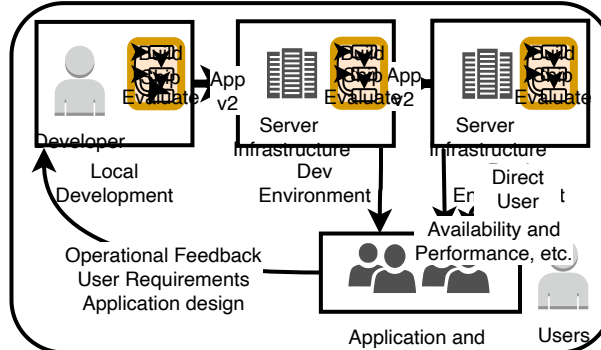


SLDC Context - your Organization/the Universe

Normal Spacetime

Your Organization

Software Development Lifecycle



SLDC of other Apps

Org policy, procedures

Other initiatives, opportunities

Everything Else

Agenda

- ✓ Preliminaries
- ✓ A General Model
- ➡ Catalog of Practices
 - ➡ Build
 - Ship
 - Evaluate
 - Example Workflows
 - Considerations, Caveats, and Advice

The Build Step

- Usually executes on a CI server or Repository service
- With a nice IDE, it can start while the code is still being written
- locally at commit time

Build Practices

- static analysis: style linting, error detection, automated refactoring
- dependency resolution
- vulnerability checking
- execute unit tests
- manual code review
- semantic versioning
- packaging: eggs, wheels, docker containers, golden VM images
- distribution: upload the release/artifact to a package service or artifact store

Agenda

- ✓ Preliminaries
- ✓ A General Model
- ➡ Catalog of Practices
 - ✓ Build
 - ➡ Ship
 - Evaluate
- Example Workflows
- Considerations, Caveats, and Advice

The Ship Step

Can execute from:

- a CI server
- repository service
- configuration management service
- an infrastructure provider
- or locally

Ship Step Practices

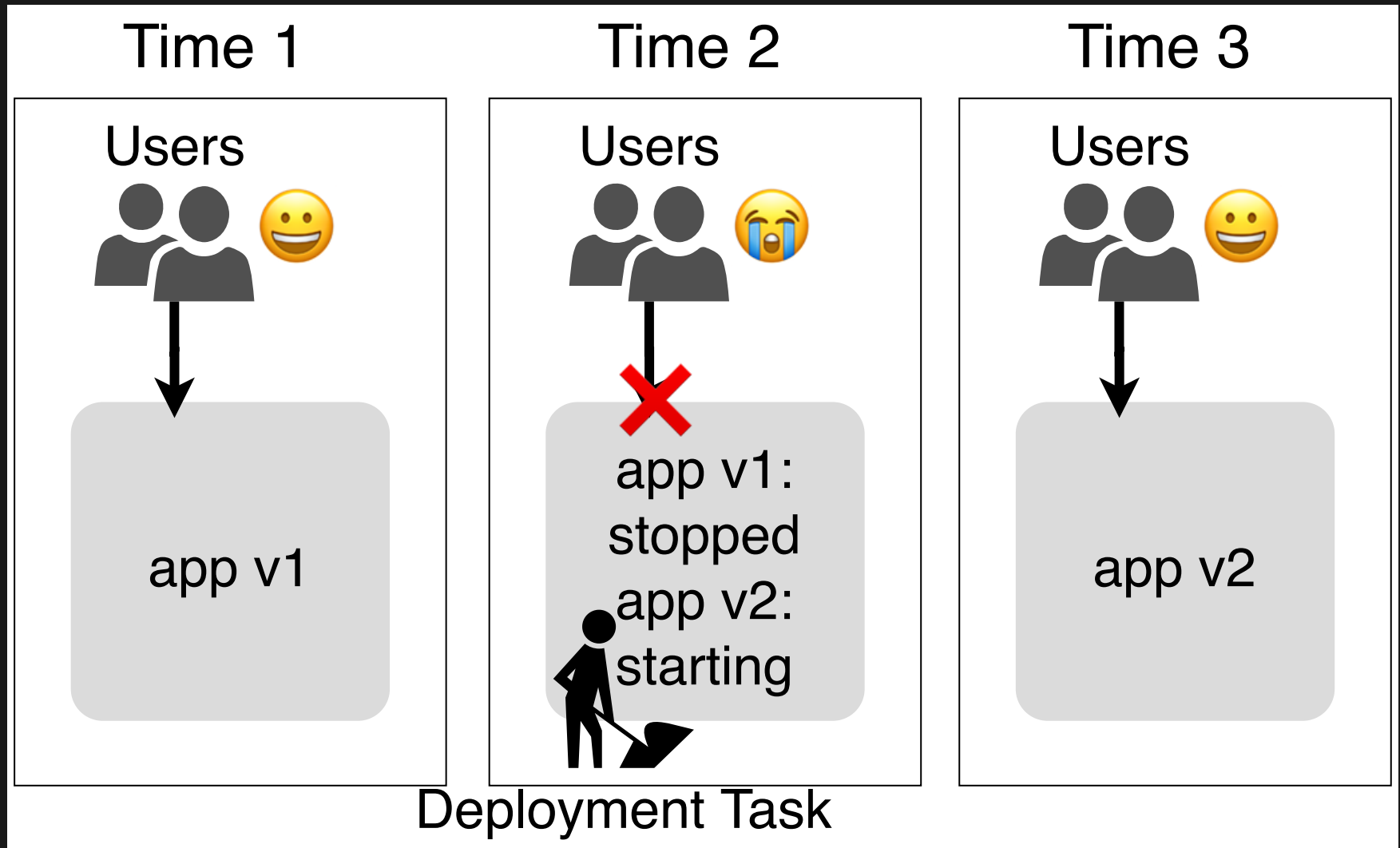
Can Include:

- environment creation
- infrastructure preparations
- configuration injection
- launch of a release/artifact
- new instance health checks
- directing traffic to the release/artifact
- rollback to previous release/artifact

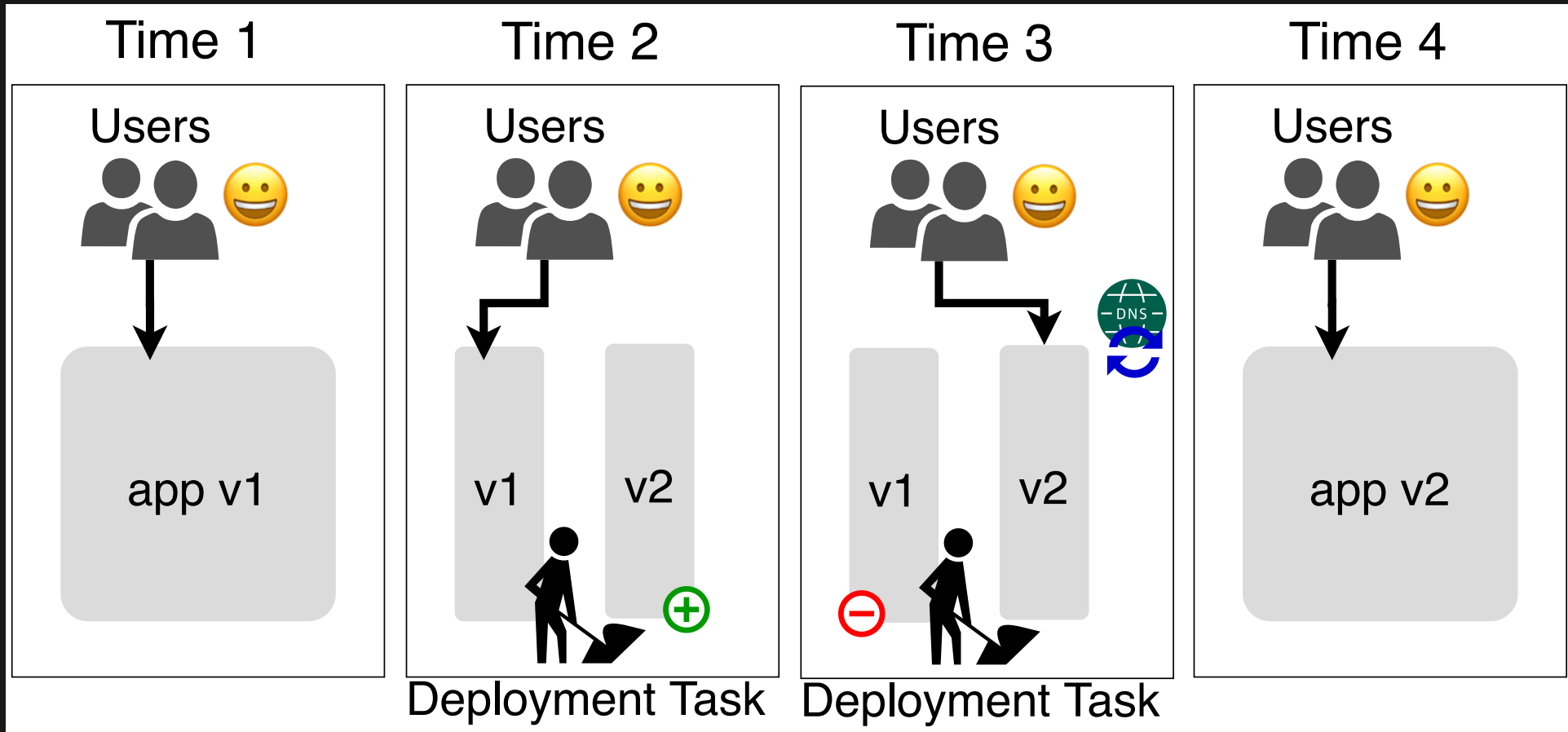
Ship Step - Directing Traffic

There are a lot of ways to do this.

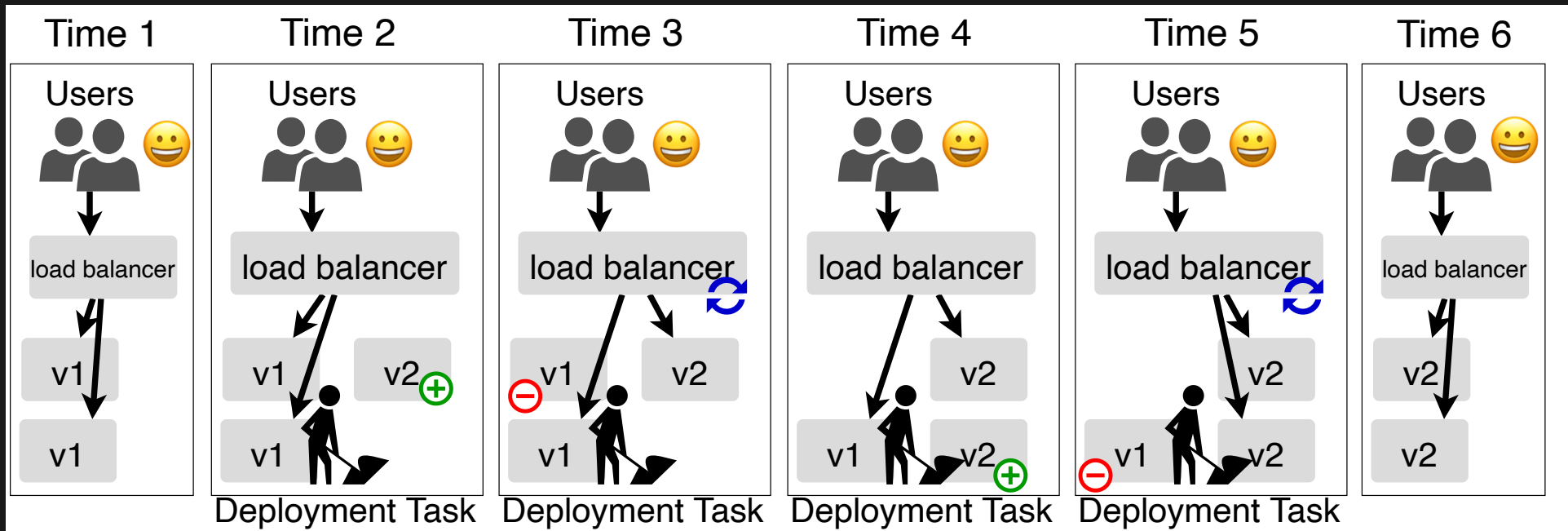
Directing Traffic - Live server



Directing Traffic - IP/DNS swap

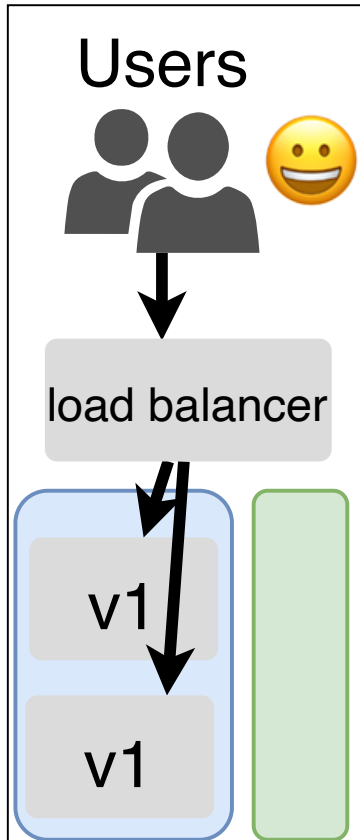


Directing Traffic - Rolling Update

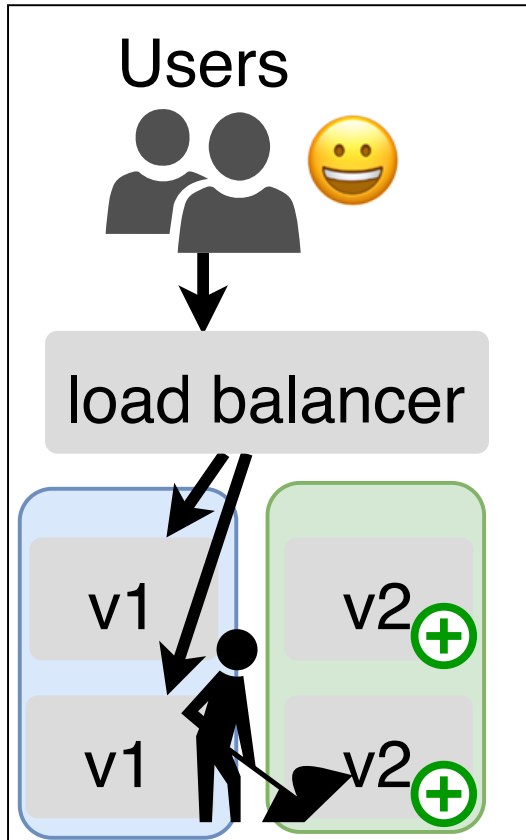


Directing Traffic - Blue/Green

Time 1

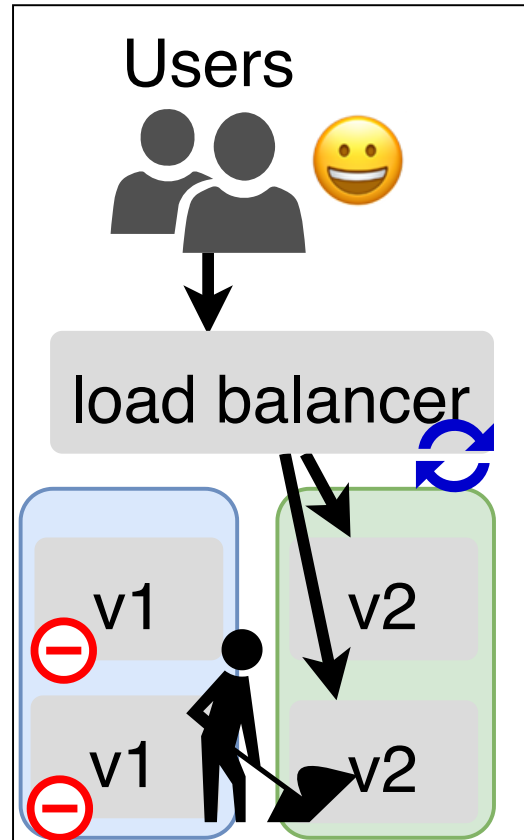


Time 2



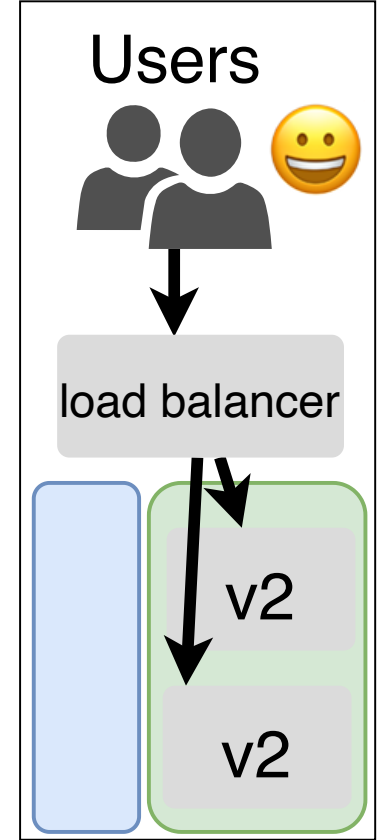
Deployment Task

Time 3

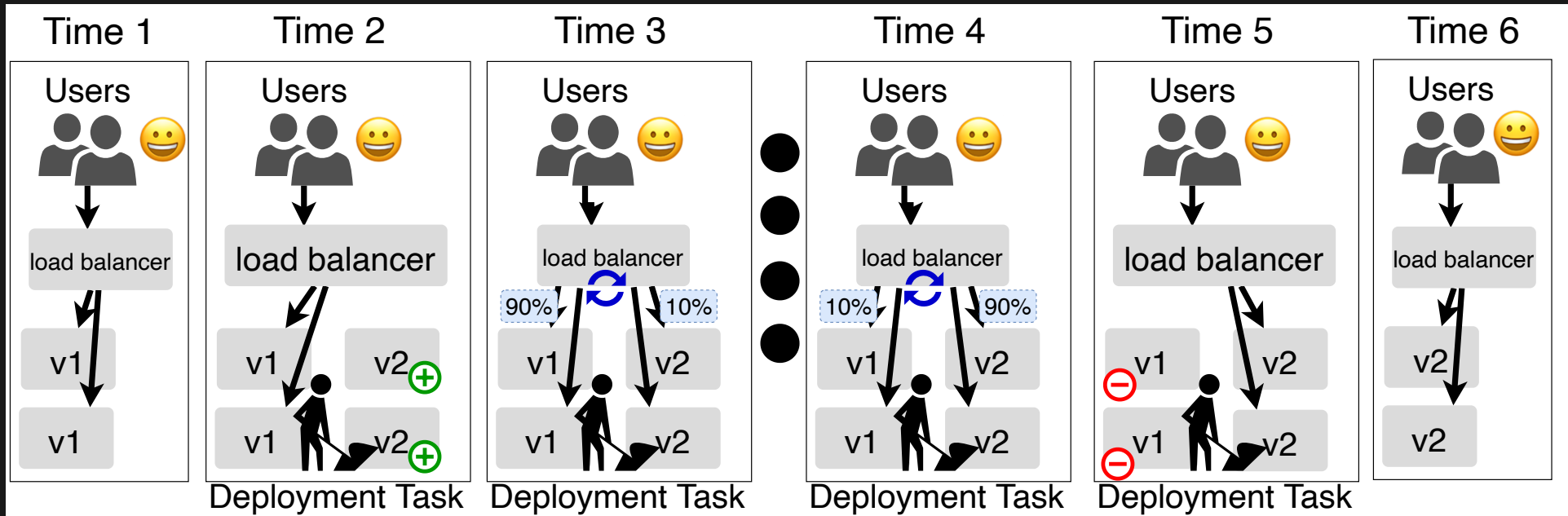


Deployment Task

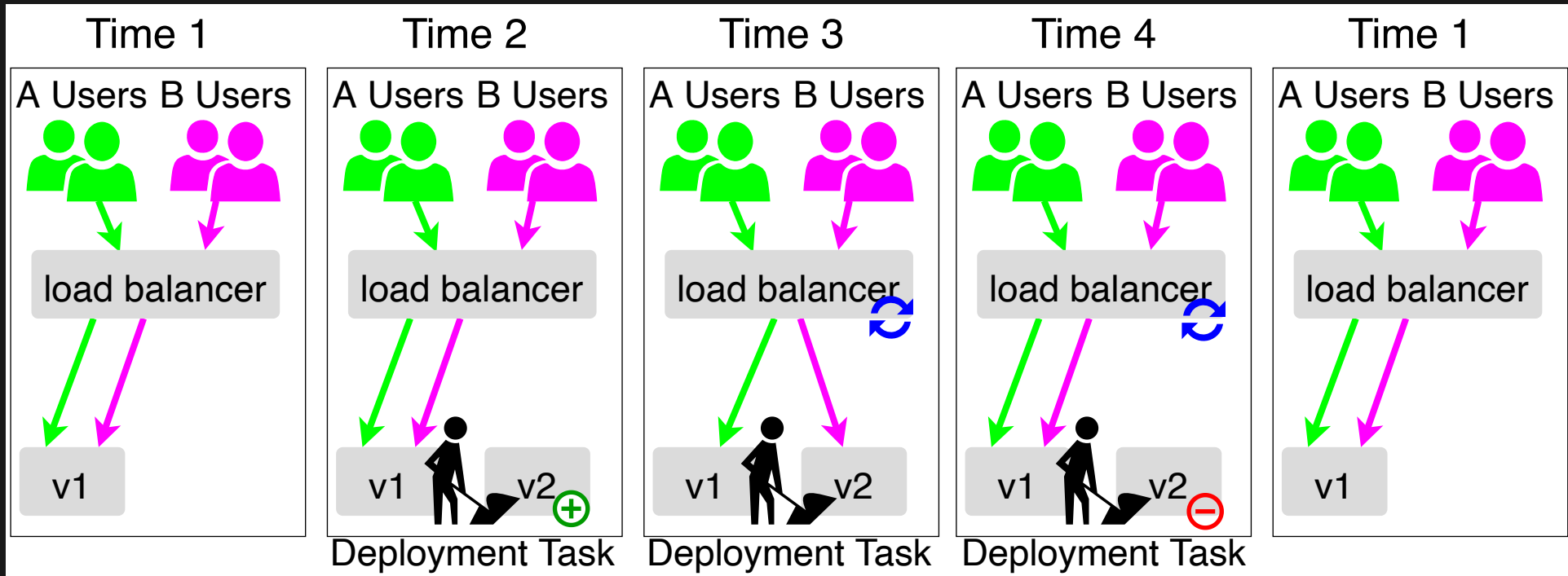
Time 4



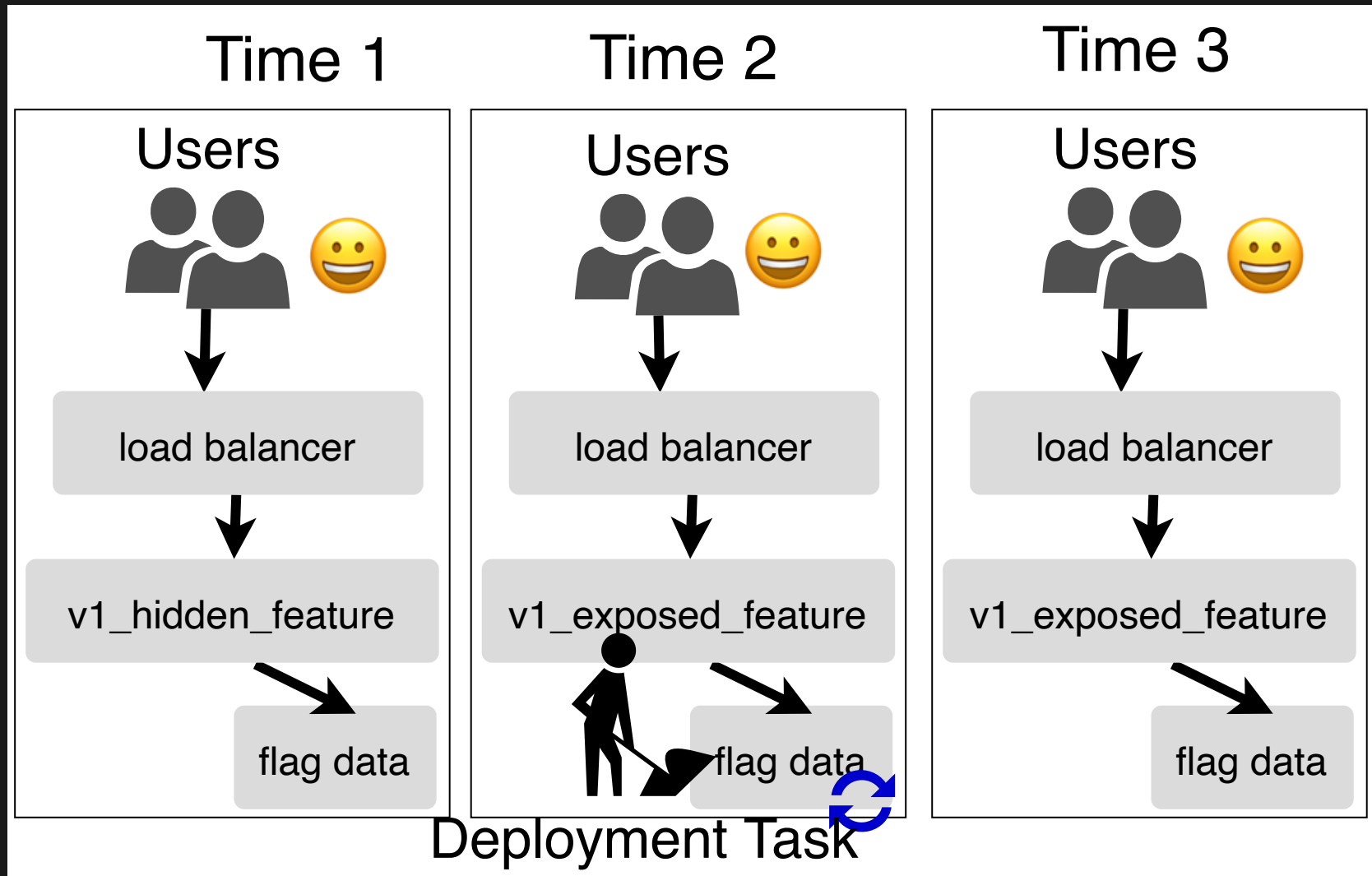
Directing Traffic - Canary



Directing Traffic - A/B Testing



Directing Traffic - Feature Flags



Agenda

- ✓ Preliminaries
- ✓ A General Model
- ➡ Catalog of Practices
 - ✓ Build
 - ✓ Ship
 - ➡ Evaluate
- Example Workflows
- Considerations, Caveats, and Advice

The Evaluate Step

Can execute from:

- a CI server
- in the brains of stakeholders
- or other external systems

Evaluate Step Practices

- execute integration or functional tests
- run security assessments
- seek user feedback
- collect availability and performance metrics
- track Key Performance Indicators
- communicate what was learned

Evaluate Step - Communication

Information may be relevant to the next:

- deployment workflow step
- development iteration
- operations/infrastructure change cycle
- organizational strategic plan

Agenda

- ✓ Preliminaries
- ✓ A General Model
- ✓ Catalog of Practices
- ➡ Example Workflows
 - Considerations, Caveats, and Advice

Microservices Entrant

Startup using a popular container platform with hopes of fame and fortune

Deployment in_development:

Build:

- lint and syntax check locally with IDE
- unit tests locally with IDE for test driven development
- docker build

Ship:

- docker-compose up, with the "live server" pattern

Evaluate:

- subset of integration tests with mocked api calls
- feedback from testing determines Go/NoGo for Deployment

test_env

Deployment test_env:

Build:

Weekend Warrior

Uses a Platform as a Service to host hobby/side project

Deployment in_development:

Build:

- lint and syntax check locally with IDE
- docker build

Ship:

- docker-compose up, with the "live server" pattern

Evaluate:

- manual testing with local address:port
- feedback from testing determines Go/NoGo for Deployment

in_PaaS

Deployment in_PaaS:

Build:

git commit

Unseen University

Has multiple sites that use a content management system requiring database connections, file storage, and custom compilation

Deployment in_development:

Build:

- lint and syntax check locally with IDE
- VMs provided with repo checkout via vagrant shared f

Ship:

- vagrant up, creates a disposable environment
- vagrant kicks off saltstack configuration management

initializes database, loadbalancer, executes the CMS
configuration and compilation script, and starts the servi

Evaluate:

- manual testing with VM address:port
- feedback from testing determines Go/NoGo for Deploym

datacenter_dev_env

Agenda

- ✓ Preliminaries
- ✓ A General Model
- ✓ Catalog of Practices
- ✓ Example Workflows
- ➔ Considerations, Caveats, and Advice

Getting Value from Deployment

A deployment workflow can deliver:

- speed
- safety
- ease of use
- security assurances
- reduce risk
- revert capability
- auditability
- consistency

But, it takes time, effort, and constant vigilance

Maturity Models

The Continuous Delivery Maturity Model					
	Base	Beginner	Intermediate	Advanced	Expert
Culture & Organisation	<ul style="list-style-type: none"> • Prioritised work • Defined and documented process • Frequent commits 	<ul style="list-style-type: none"> • One backlog per team • Show the path • Stable teams • Adopt basic Agile methods • Remove boundary dev & test 	<ul style="list-style-type: none"> • Extended team collaboration • Component ownership • Act on metrics • Remove boundary dev & ops • Common process for all changes • Decentralised decisions 	<ul style="list-style-type: none"> • Dedicated solo team • Team responsible all the way to prod • Deploy disconnected from Release • Continuous improvement (Kaizen) 	<ul style="list-style-type: none"> • Cross functional teams • No nitbicks (always nit forward)
Design & Architecture	<ul style="list-style-type: none"> • Consolidated platform & technology 	<ul style="list-style-type: none"> • Organise system into modules • API management • Library management • Version control DB changes 	<ul style="list-style-type: none"> • No (or minimal) branching • Branch by abstraction • Configuration as code • Feature toggling • Making components out of modules 	<ul style="list-style-type: none"> • Full component based architecture • Push business metrics 	<ul style="list-style-type: none"> • Infrastructure as code
Build & Deploy	<ul style="list-style-type: none"> • Versioned code base • Scripted builds • Basic scheduled builds (CI) • Dedicated build server • Documented manual deploy • Some deployment scripts exist 	<ul style="list-style-type: none"> • Pulling builds • Builds are stored • Manual log & monitoring • First step towards standardised deploys 	<ul style="list-style-type: none"> • Auto triggered builds (commit hooks) • Automated log & monitoring • Build once deploy anywhere • Automated bulk of DB changes • Basic pipeline with deploys to prod • Sorted config changes (e.g. app server) • Standard process for all environments 	<ul style="list-style-type: none"> • Zero downtime deploys • Multiple build machines • Full automatic DB deploys 	<ul style="list-style-type: none"> • Build backup • Zero touch continuous deployments
Test & Verification	<ul style="list-style-type: none"> • Automatic unit tests • Separate test environment 	<ul style="list-style-type: none"> • Automatic integration tests 	<ul style="list-style-type: none"> • Automatic component tests (isolated) • Some automatic acceptance tests 	<ul style="list-style-type: none"> • Full automatic acceptance tests • Automatic performance tests • Automatic security tests • Risk based manual testing 	<ul style="list-style-type: none"> • Verify expected business value
Information Reporting	<ul style="list-style-type: none"> • Baseline process metrics • Manual reporting 	<ul style="list-style-type: none"> • Measure the process • Static code analysis • Scheduled quality reports 	<ul style="list-style-type: none"> • Common information model • Traceability built into pipeline • Report history is available 	<ul style="list-style-type: none"> • Graphing as a service • Dynamic test coverage analysis • Automatic security tests • Repetition analysis 	<ul style="list-style-type: none"> • Dynamic graphing and dashboards • Cross site analysis

Are not qualified to judge you

Assess Your Needs

- Who are you?
- What does your application mean to you?
- What capabilities are practical to implement?

Don't forget

- avoid stale data
- accomodate cache warmup times
- look for partially or undocumented dependencies
- don't make incompatible database changes
- prevent cache poisoning
- monitor for runaway processes
- get stakeholder buy-in

Also

I believe in you!

- get started with the easy stuff
- notice repetitive tasks and reoccurring issues
- you can stop when you have what you need
- use, share, and improve community tools

Agenda

All done!

- ✓ Preliminaries
- ✓ A General Model
- ✓ Catalog of Practices
- ✓ Example Workflows
- ✓ Considerations, Caveats, and Advice

Resources

- Full Stack Python - Deployment: <https://www.fullstackpython.com/deployment.html>
- Systems development life cycle: https://en.wikipedia.org/wiki/Systems_development_life_cycle
- Release management: https://en.wikipedia.org/wiki/Release_management
- Six Strategies for Application Deployment: <https://thenewstack.io/deployment-strategies/>
- Semantic Versioning: <https://semver.org/>
- The Visible Ops Handbook: <https://itpi.org/the-visible-ops-book-series/visible-ops-handbook-review/>
- The Practice of Cloud System Administration: DevOps and SRE Practices for Web Services: <https://everythingsysadmin.com/books.html>
- Mykel Alvis - Developing for Deterministic Deliveries, the live version: <https://www.youtube.com/watch?v=gTa1fJuPP0E&start=616>
- Ansible - Playbook Example: Continuous Delivery and Rolling Upgrades: https://docs.ansible.com/ansible/latest/user_guide/guide_rolling_upgrade.html
- System Deployment Tips and Techniques [httphttps://www.ambysoft.com/essays/deploymentTips.html](https://www.ambysoft.com/essays/deploymentTips.html)
- USGS - Deployment Best Practices: <https://www.usgs.gov/products/software/software-management/deployment-best-practices>
- Kubernetes for Sysadmins – Kelsey Hightower at PuppetConf 2016: <https://www.youtube.com/watch?v=HlAXp0-M6SY>
- Continuous Integration & Delivery (CI/CD) for Kubernetes Using CircleCI & Helm: <https://medium.com/velotio-perspectives/continuous-integration-delivery-ci-cd-for-kubernetes-using-circleci-helm-b8b0a91ef1a3>
- Feature Toggles (aka Feature Flags): <https://martinfowler.com/articles/feature-toggles.html>
- Matching Supply with Demand: An Introduction to Operations Management <http://cachon-terwiesch.net/3e/>
- Principles behind the Agile Manifesto: <https://agilemanifesto.org/principles.html>
- The Twelve-Factor App: <https://12factor.net/>
- Cover image: <https://www.usgs.gov/media/images/cartographers-field>
- Maturity Model: <https://www.infoq.com/articles/Continuous-Delivery-Maturity-Model/>