Front Matters

- Amazon AWS Key Issues
  - Don’t commit secret keys to GitHub
  - You can pass these to Travis-CI, Code Deploy using environment variables.
  - How many of you saw the Canvas announcements about this?
- Submission template is available from http://courses.airavata.org/projects.html
- I will extend Project Milestone 1’s deadline until Monday, September 26th
  - Look for updates from Canvas
- We will have another help-a-thon on Thursday, September 22nd during regular class hours
- Note different office hours location
- Next week: two guest lectures about two science gateways
  - Attendance will be taken

Continuous Integration and Deployment

Applying to Microservices

Assumptions

- You know what your dependencies are for your services
  - Libraries, run time environments, compilers, etc.
- You have build systems that can build and test your services.
  - Gradle, Maven, Ant, etc are some popular choices
  - These help you organize your projects, too
- You have different branches for your code in GitHub to separate features, integration code, and release code.
- CI applies to only selected branches
  - Test/Integration Branch
    - We called this Dev in earlier lectures
  - Release Branch

Continuous Integration Cycle

CI is distributed building and testing for multi-developer teams.

Continuous Integration

- Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day.
- Each check-in is then verified by an automated build, allowing teams to detect problems early.
- By integrating regularly, you can detect errors quickly, and locate them more easily.

https://www.thoughtworks.com/continuous-integration

Continuous Integration Practices

- Maintain a single source repository
- You have a defined build process
- Automate the build
- Make your build self-testing
- Every commit to the “integration” branch should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what’s happening
- Automate deployment

https://www.thoughtworks.com/continuous-integration

CI Sequence

- Developers check out code into their private workspaces.
- When done, commit the changes to the repository.
- The CI server monitors the repository and checks out changes when they occur.
- The CI server builds the system and runs unit and integration tests.
- The CI server releases deployable artefacts for testing.
- The CI server assigns a build label to the version of the code it just built.
- The CI server informs the team of the successful build.
- If the build or tests fail, the CI server alerts the team.
- The team fix the issue at the earliest opportunity.
- Continue to continually integrate and test throughout the project.

https://www.thoughtworks.com/continuous-integration

CI Do's and Don'ts

- Do check in frequently
- Don't check in broken code
- Don't check in untested code
- Don't check in when the build is broken
- Don't go home after checking in until the system builds

https://www.thoughtworks.com/continuous-integration

Does CI Really Work?

- You will find some sceptics.
- Greatest Sin: Ignoring broken builds and associated nag emails
- Main issue: feature branches with major new developments don’t fit easily into the “commit and build often” and “don’t break the build” requirements.
- It takes some thought and planning
  - Abstractions and mock interfaces for new features
  - Test Driven Design
- Development environments need to look like production environments
  - Vagrant, Docker, etc are tools that help here.

Continuous Integration Tools: Apache Jenkins

- Apache Jenkins
  - Closely associated with Java projects and build systems
- See example: https://github.com/airavata-courses/TeamAlpha/wiki/5.-Project-Milestone-5:-Details-and-Instructions
- Jenkins is an open source tool for automating code builds.
- Jenkins 1.x isn't scriptable.
  - Everything goes through the user interface
  - This is changing in Jenkins 2.
  - IMHO: this is an essential feature

Reasons to Use Jenkins

- Works well with Java-centric projects
  - Good support for Maven, Ant build systems
- You can make it do lots of other stuff with scripts in pre- and post-build stages.
  - Deploy databases
  - Run code
- You want to run your own CI system
  - Not shared like Travis-CI

Continuous Integration with Travis-CI

- Hosted service that builds codes on Ubuntu Linux flavors
  - Uses apt tools to install
- Build process is controlled using a YAML file
  - .travis.yml
- Programming language agnostic
- Built in tools for many common operations
  - But you can ultimately do almost any thing as root

.travis.yml file

- Put this in the repository that you want Travis to build.
- Travis.yml lets you specify things like
  - What programming language your project uses
  - What commands or scripts you want to be executed before each build (for example, to install or clone your project's dependencies)
  - What command is used to run your test suite
  - Emails, Campfire and IRC rooms to notify about build failures

Travis-CI Phase	Description
before_install	Run any custom commands before installation
install	Set up your environment: install compilers and runtimes, databases, services, etc. You can use packaged services or specify things manually
before_script	Run any custom scripts before building your code.
script	Build and test your code here
after_success, after_failure	Steps to take after the script phase, such as notifications, alternate paths
before_deploy, deploy, after_deploy (optional)	Integrate your successful builds with a suported continuous deployment providers like AWS CodeDeploy
after_script	Clean up

Some .travis.yml Examples

- https://github.com/airavata-courses/TeamApex/blob/master/.travis.yml
- https://github.com/marpierc/CodeDeployTest/blob/master/.travis.yml
  - Not advisable to use but shows how you can really shove a lot of stuff into Travis.

CI and Microservices

- Remember that each service is a separate OS process.
  - Can be deployed on a separate VM or container per service
  - We'll look at some of these patterns in future lectures
- How do you handle this with a CI system like Travis?
  - Your time to speak up
  - Hint: you may need to make multiple branches in GitHub
  - Hint: can you do all your builds in one call to Travis?
    - Or do you need to to make separate calls to Travis for each service?

Testing Microservices

- You want to do this during the CI phase and the CD phase
  - CI->”Develop” branch(es) of your repo
  - CD->”Release” branch(es) of your repo
- For the CI phase, you have at least two choices on how to deploy microservices for testing
  - **What are they?**
  - (See next slides)

Choice #1: Build All Services

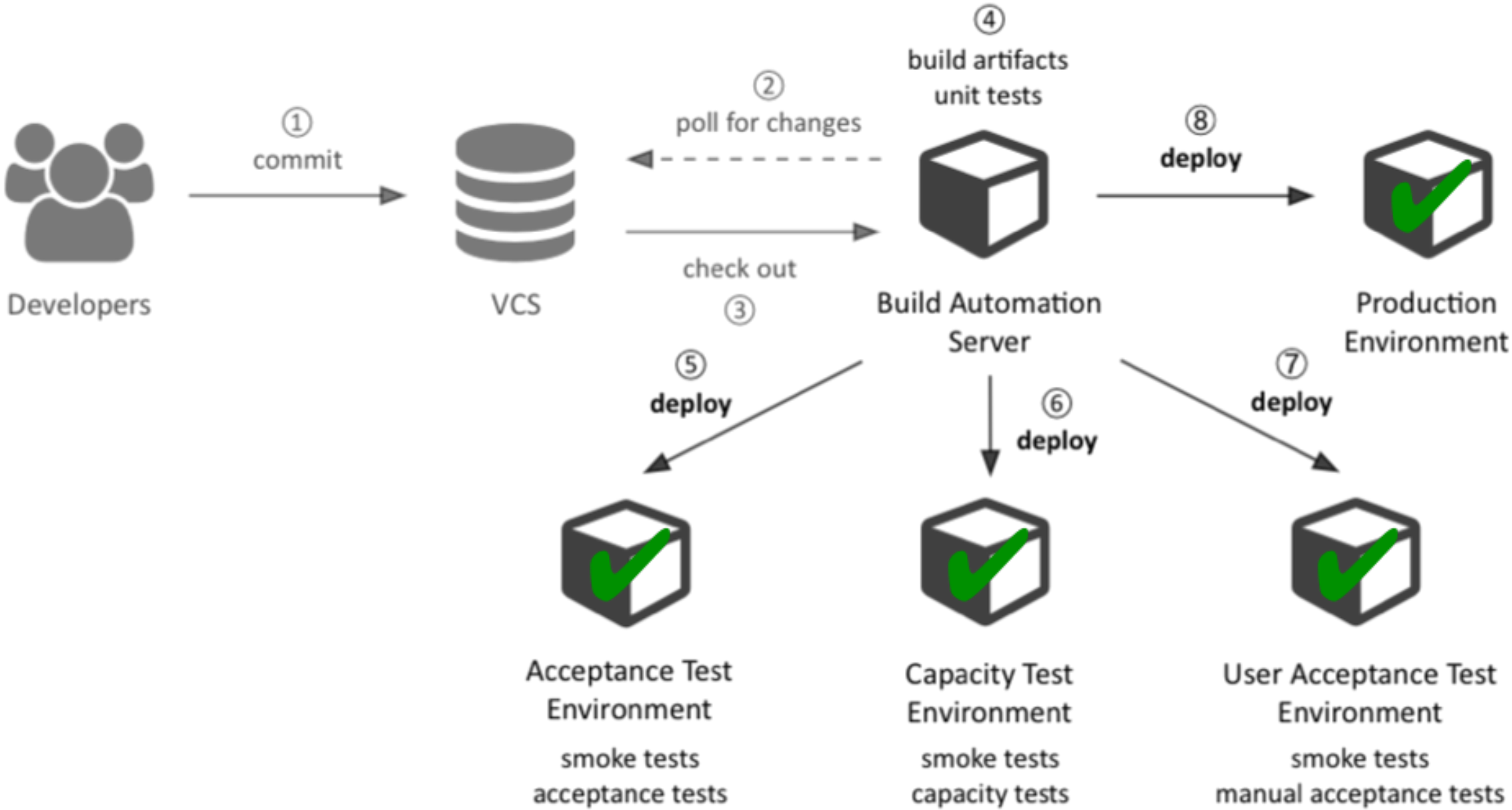
- Build and test the whole environment on Travis-CI
  - Your current check-in plus all other stable “dev” branch services.
- Advantage: you can go quickly to deployment with some confidence
- Disadvantage: may not be feasible with many services, very slow builds
- What kinds of tests?
  - Unit, integration and end-to-end system tests
- For gateways, SauceLabs is an interesting tool integrated into Travis-CI.
  - Launch Selenium browser tests after your build.

Choice #2: Build and Test One Service at a Time

- Use Travis-CI to build only one service at a time, use “mock” services for the other services.
- Advantage: scales better than Choice #1 for large deployments
- Disadvantage: you’ll need full-fledged integration and deployment(s) environment before going to full deployment
  - Also, concurrency problems if lots of developers are uploading new services

Continuous Deployment

Or Is It Continuous Delivery?


④
build artifacts
unit tests
②
poll for changes
①
commit
⑧
deploy
Developers
VCS
check out
③
Build Automation
Server
Production
Environment
⑤
deploy
⑥
deploy
⑦
deploy
Acceptance Test
Environment
smoke tests
acceptance tests
Capacity Test
Environment
smoke tests
capacity tests
User Acceptance Test
Environment
smoke tests
manual acceptance tests

Continuous Deployment Is …

- CD is a lot like CI
- CI can be used in non-DevOps fashion for disciplined developer teams
  - Avoid painful integration of divergent branches.
  - You don’t need Software as a Service to use CI
- CD extends CI to frequently push working code into production.
  - Applies to Software as a Service systems
- You could use CD for end user software delivery theoretically but that takes a lot more quality assurance testing.
  - With SaaS deployments, you precisely control the deployment environments

The Golden CD Rule

Never login directly to deployment server hosts.
Always use a tool to deploy remotely.

Some Continuous Deployment Tools

Some quick looks at Amazon Code Deploy, Ansible

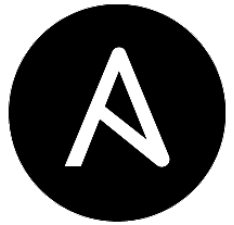
What Is Ansible?

- **Ansible** is a free-software platform for configuring and managing computers which combines multi-node software deployment, *ad hoc* task execution, and configuration management. It manages nodes over SSH or over PowerShell. Modules work over JSON and standard output and can be written in any programming language. The system uses YAML to express reusable descriptions of systems.

https://en.wikipedia.org/wiki/Ansible_(software)

Using Ansible

- You can install Ansible on your laptop/desktop, on an integration server, etc.
- For example: you could set up a UNIX cron job to use Ansible to do automated nightly integration and deployment tests.
- Personal experiences: people like Ansible
  - Get going quickly, low barrier to learn how to use it
  - Uses SSH, so nothing required to be installed or run on the target servers.
  - If you know how to write scripts and do simple system administration on Linux, you can do all of that in Ansible: not too much magic
  - Capture common tasks as playbooks
- Drawback: not as scalable as agent-based systems like Salt



ANSIBLE

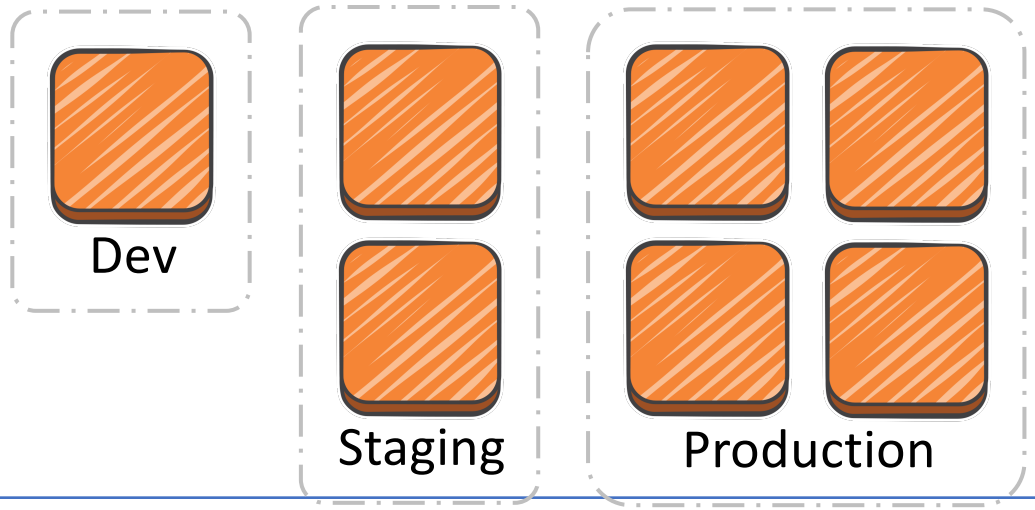
CodeDeploy

Application revisions

v1, v2, v3

CodeDeploy

Deployment groups



- Scale from 1 instance to thousands
- Deploy without downtime
- Centralize deployment control and monitoring

More Code Deploy Examples

- https://github.com/airavata-courses/TeamApex/blob/master/appspec.yml

Step 1: Package your application (with an AppSpec file)

```
version: 0.0
os: linux
files:
  - source: chef/
    destination: /etc/chef/codedeploy
  - source: target/hello.war
    destination: /var/lib/tomcat6/webapps
hooks:
  ApplicationStop:
    - location: deploy_hooks/stop-tomcat.sh
  BeforeInstall:
    - location: deploy_hooks/install-chef.sh
  AfterInstall:
    - location: deploy_hooks/librarian-install.sh
  ApplicationStart:
    - location: deploy_hooks/chef-solo.sh
  ValidateService:
    - location: deploy_hooks/verify_service.sh
```

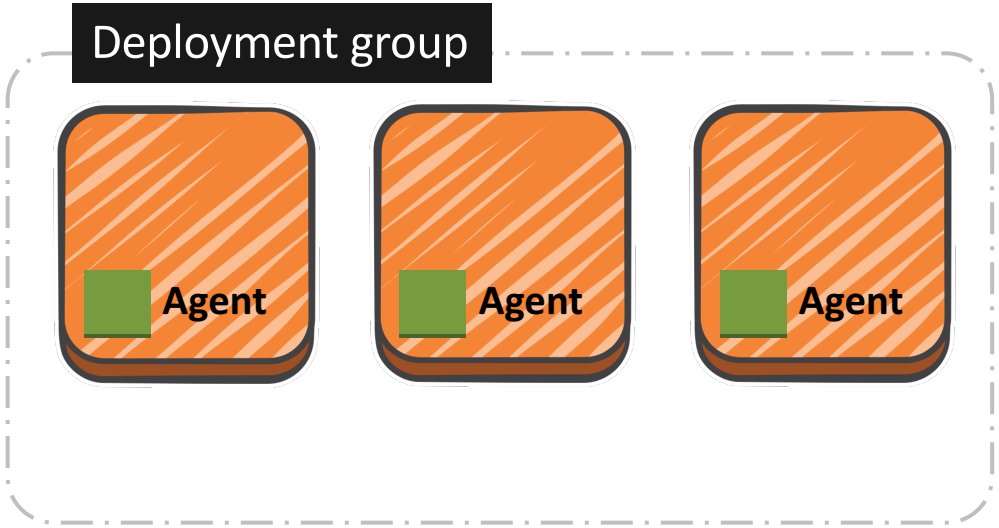
Step 1: Package your application (with an AppSpec file)

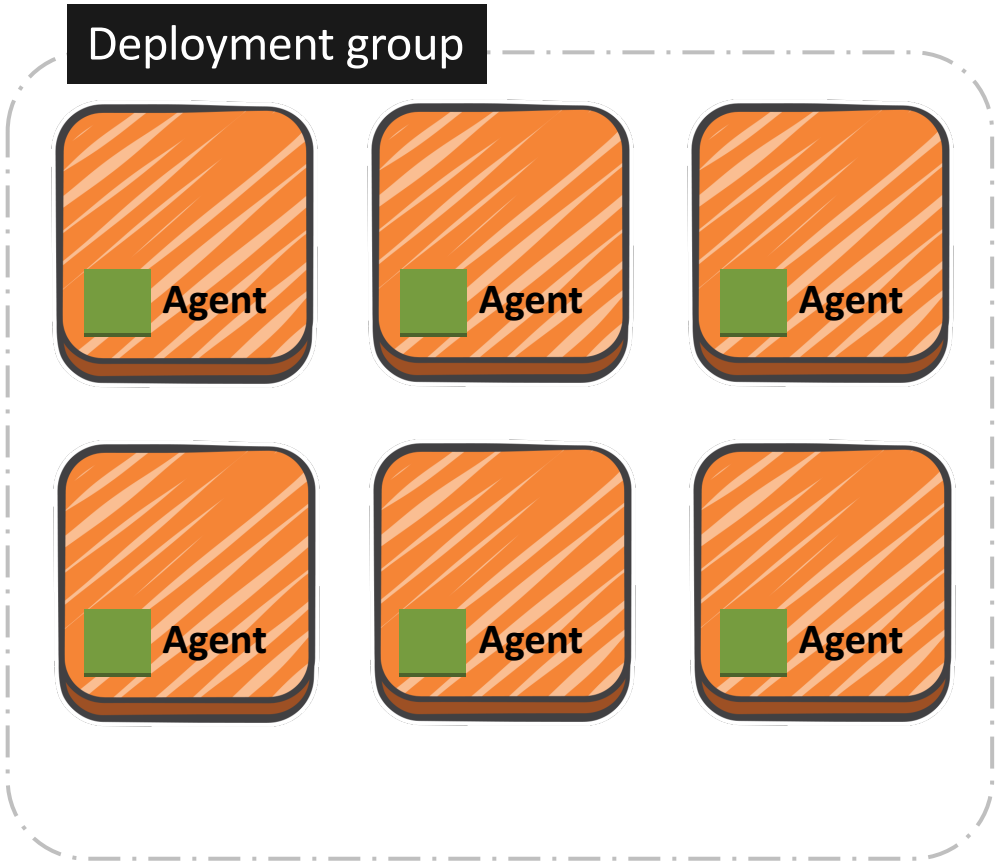
```
version: 0.0
os: linux
files:
  - source: chef/
    destination: /etc/chef/codedeploy
  - source: target/hello.war
    destination: /var/lib/tomcat6/webapps
hooks:
  ApplicationStop:
    - location: deploy_hooks/stop-tomcat.sh
  BeforeInstall:
    - location: deploy_hooks/install-chef.sh
  AfterInstall:
    - location: deploy_hooks/librarian-install.sh
  ApplicationStart:
    - location: deploy_hooks/chef-solo.sh
  ValidateService:
    - location: deploy_hooks/verify_service.sh
```

Step 1: Package your application (with an AppSpec file)

```
version: 0.0
os: linux
files:
  - source: chef/
    destination: /etc/chef/codedeploy
  - source: target/hello.war
    destination: /var/lib/tomcat6/webapps
hooks:
  ApplicationStop:
    - location: deploy_hooks/stop-tomcat.sh
  BeforeInstall:
    - location: deploy_hooks/install-chef.sh
  AfterInstall:
    - location: deploy_hooks/librarian-install.sh
  ApplicationStart:
    - location: deploy_hooks/chef-solo.sh
  ValidateService:
    - location: deploy_hooks/verify_service.sh
```

Step 2: Set up target environments





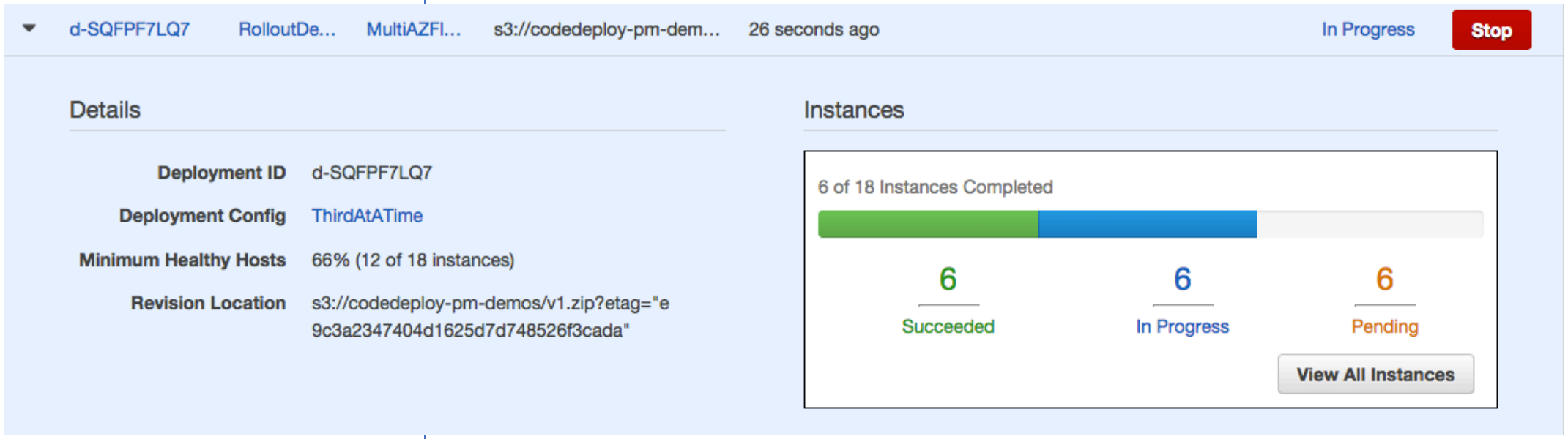
Group instances by:

- Auto Scaling group
- Amazon EC2 tag
- On-premises tag

Step 3: Deploy!

AWS CLI & SDKs
AWS Console
CI / CD Partners
GitHub

```
aws deploy create-deployment  \
--application-name MyApp  \
--deployment-group-name TargetGroup  \
--s3-location bucket=MyBucket,key=MyApp.zip
```



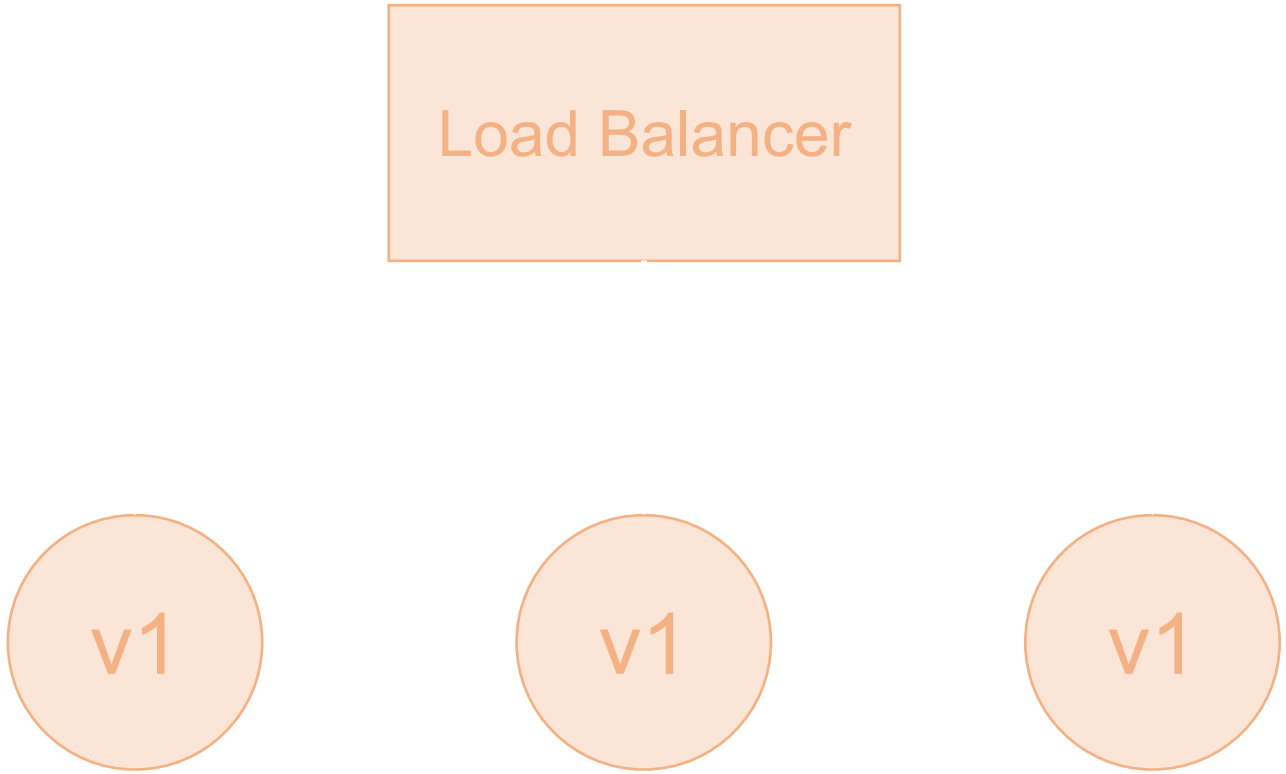
Deployment config – Choose speed

One at a time

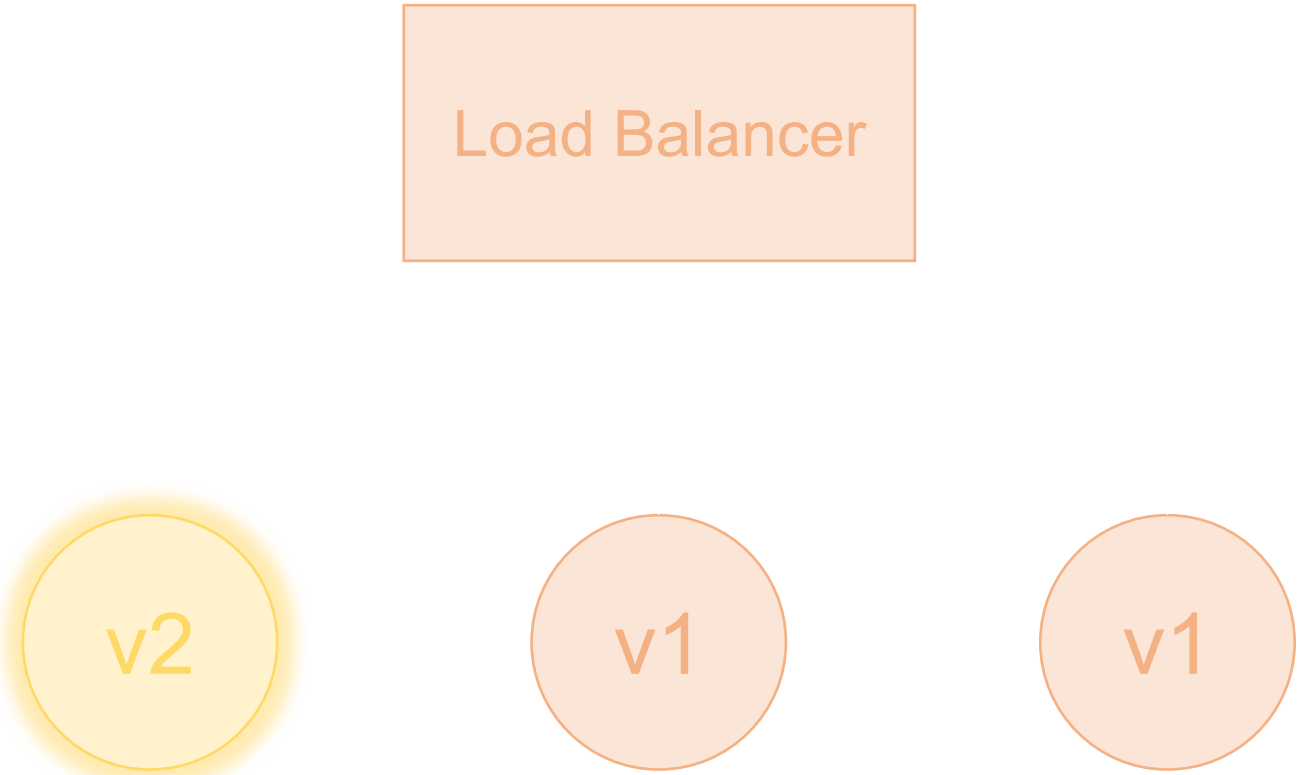
Half at a time

All at once

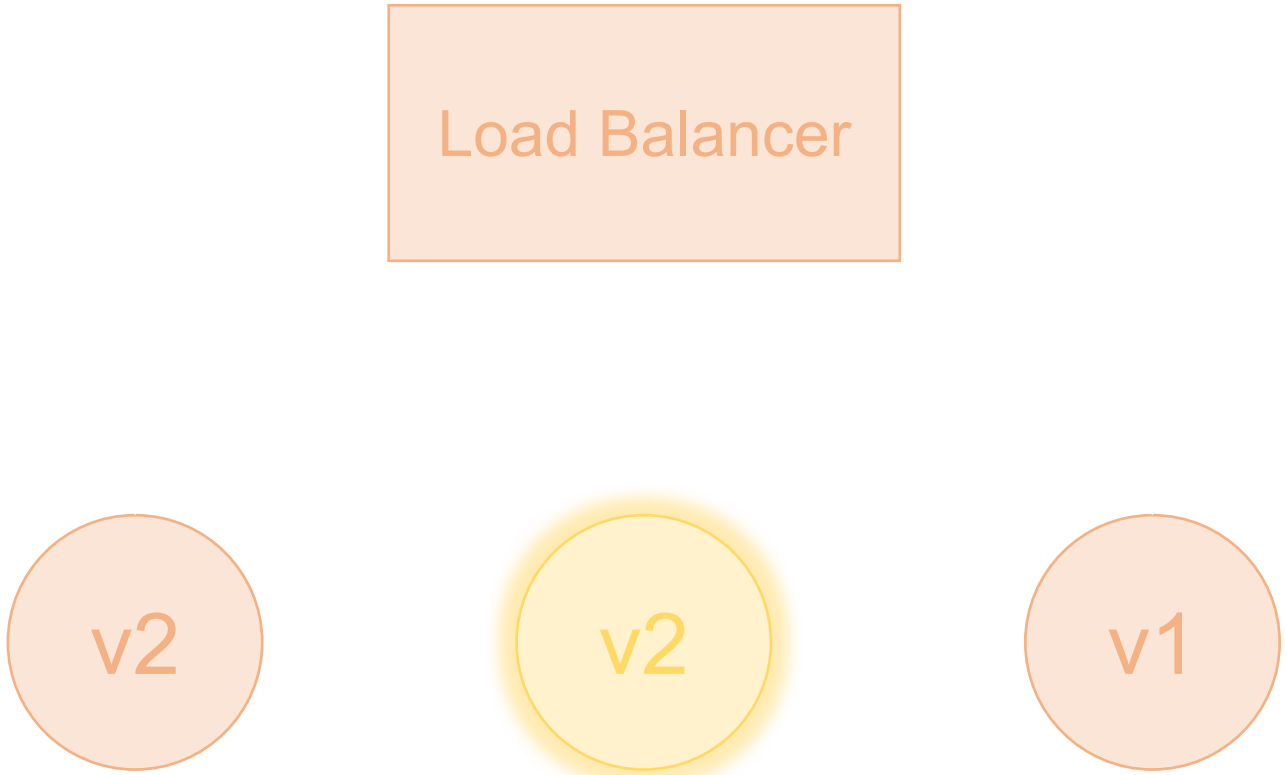
Rolling update – Deploy without downtime



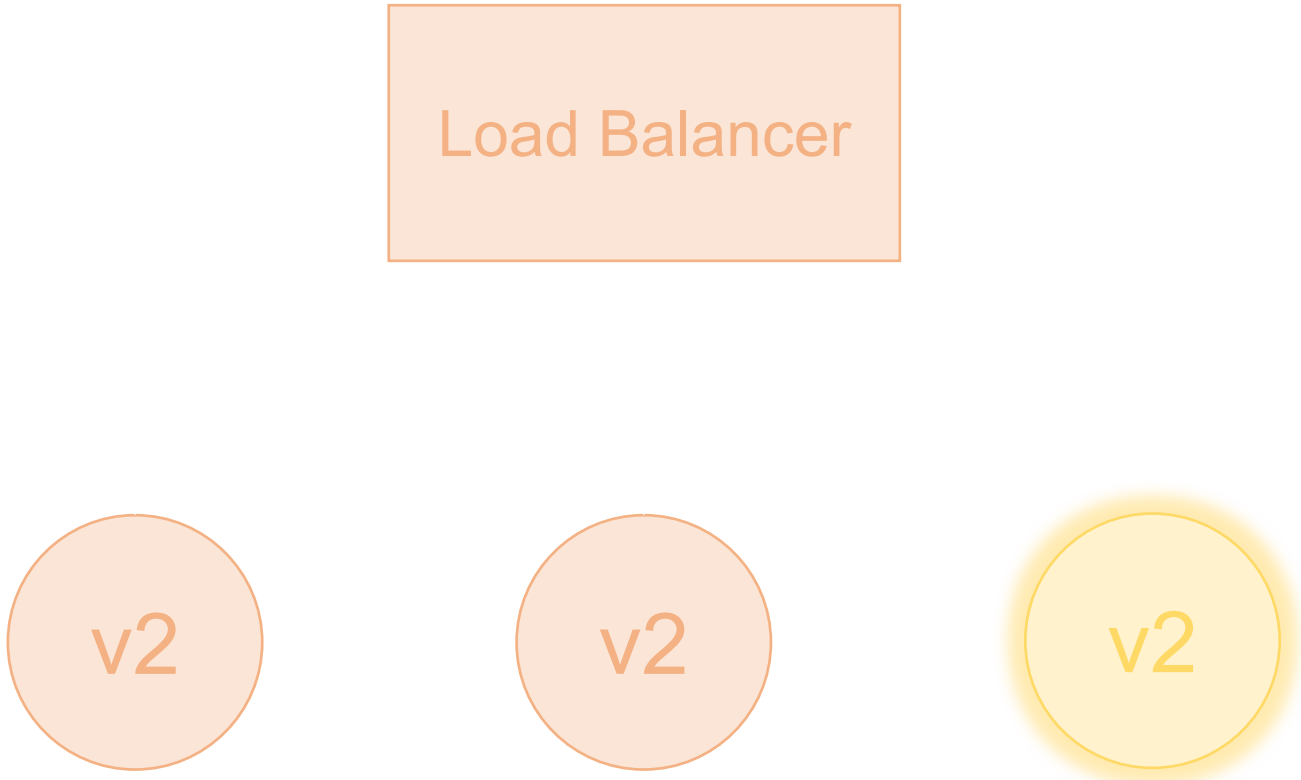
Rolling update – Deploy without downtime



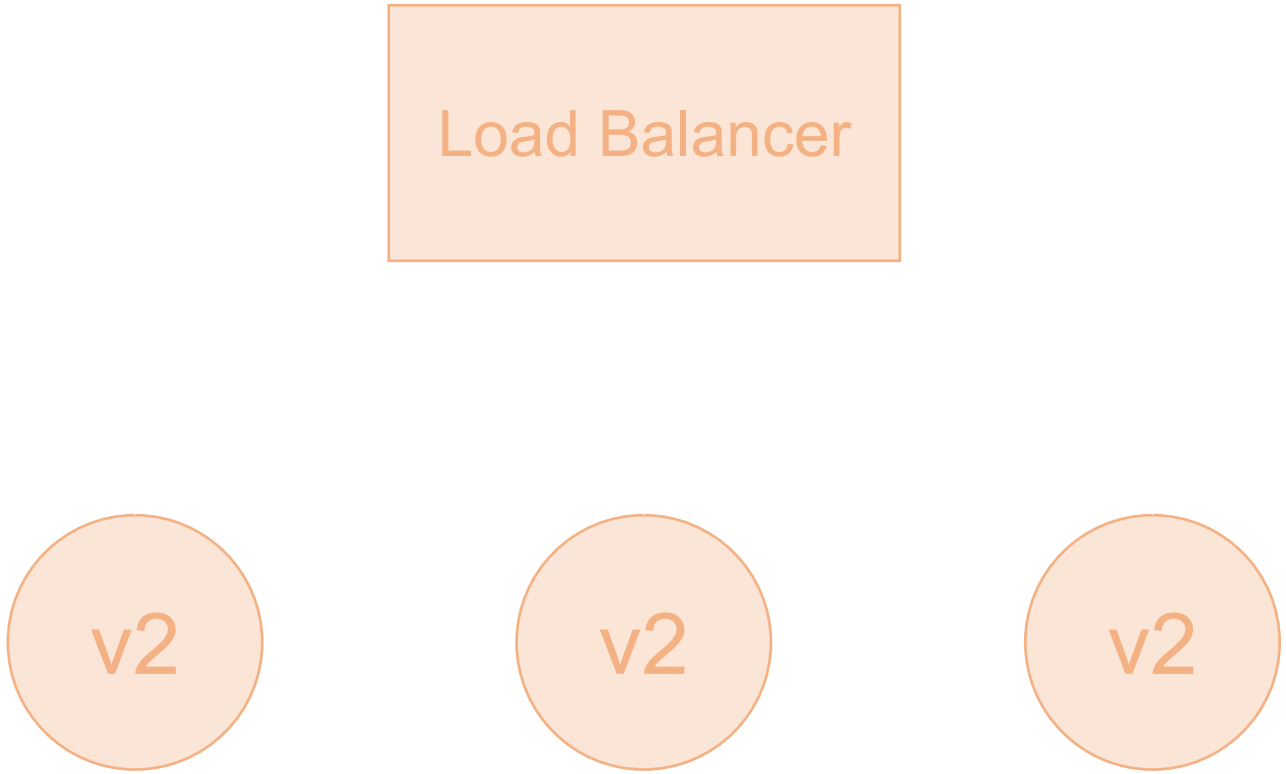
Rolling update – Deploy without downtime



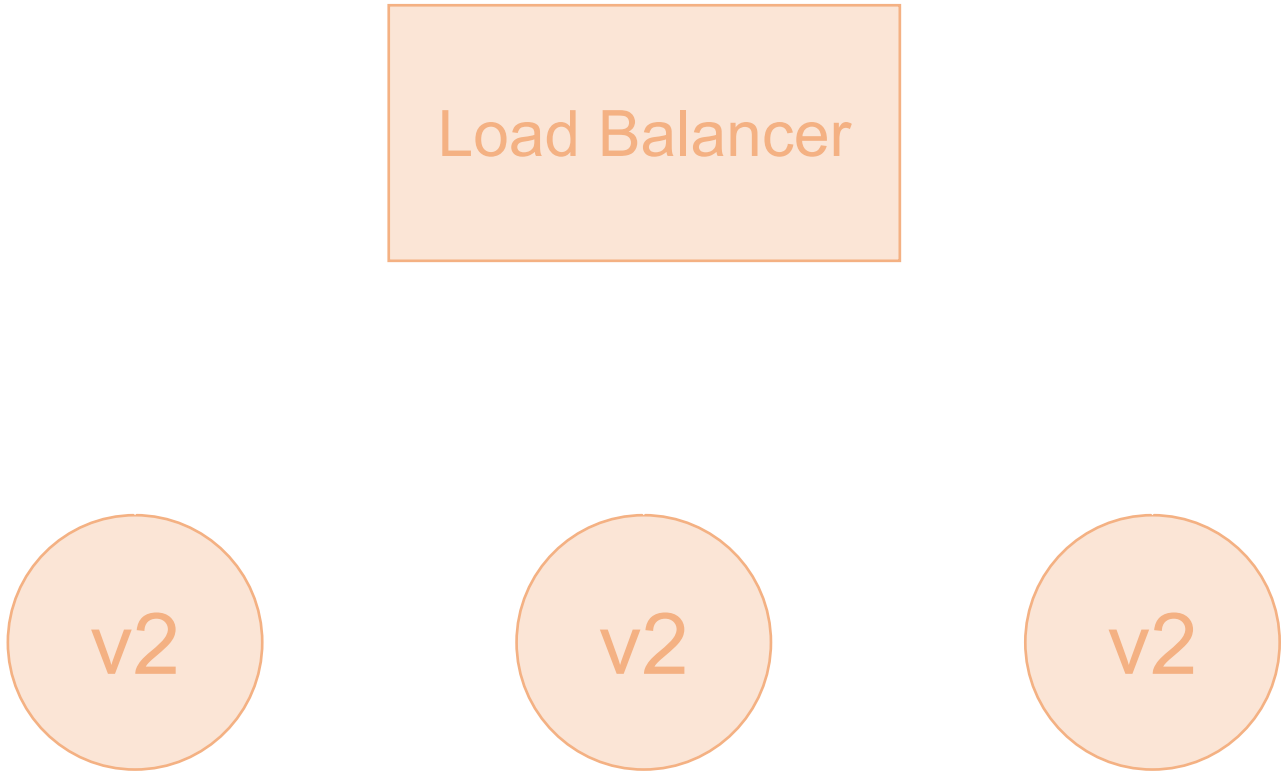
Rolling update – Deploy without downtime



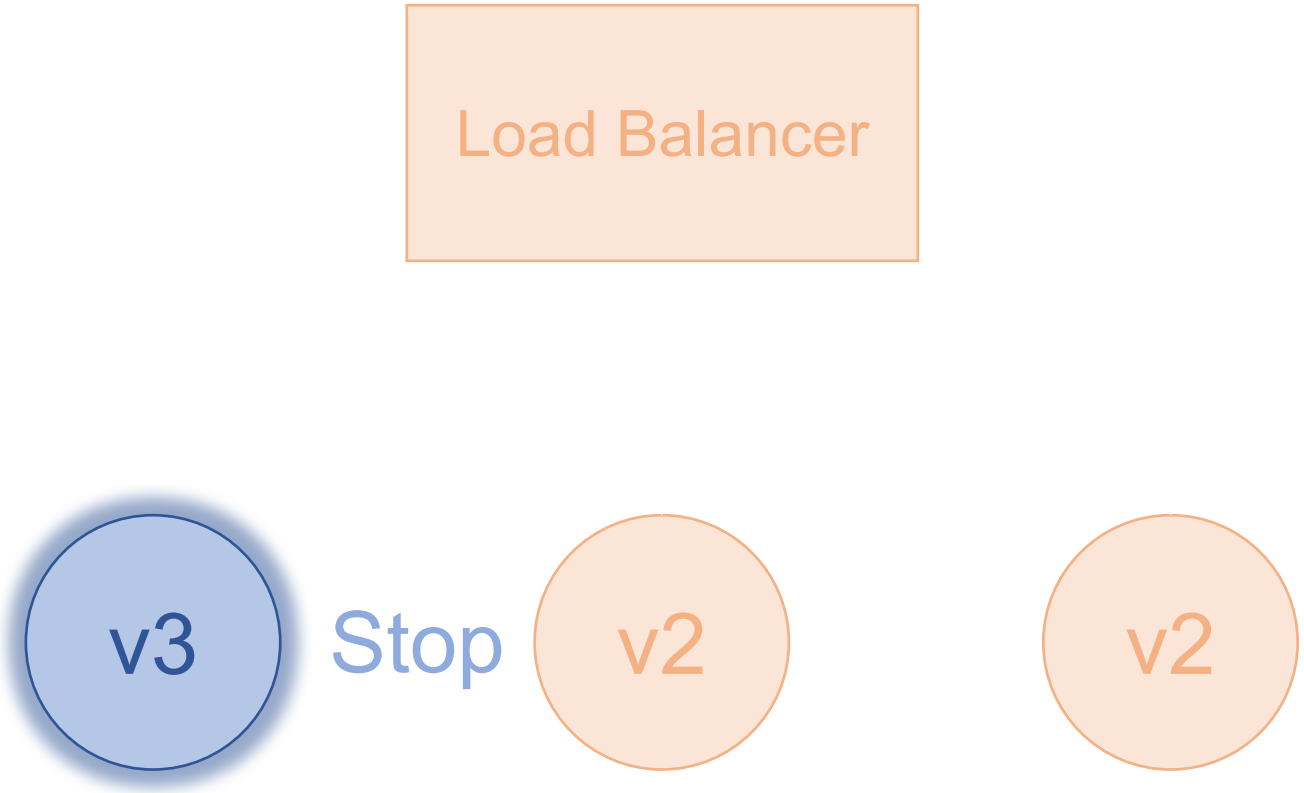
Rolling update – Deploy without downtime



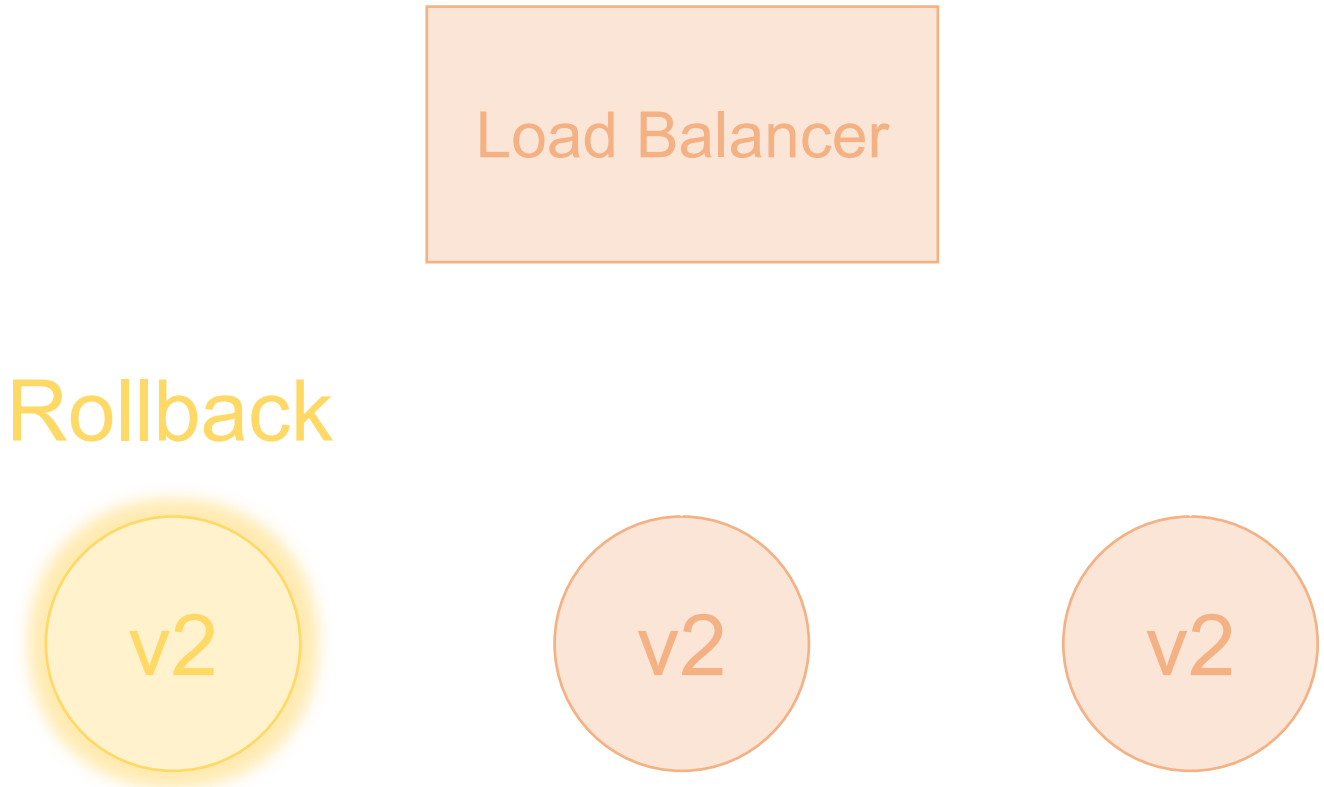
Health Tracking – Catch deployment problems



Health tracking – Catch deployment problems



Health tracking – Catch deployment problems



Health tracking – Catch deployment problems