Deploy, Manage, and Scale Your Apps with AWS Elastic Beanstalk

Andreas Chatzakis, Solutions Architect

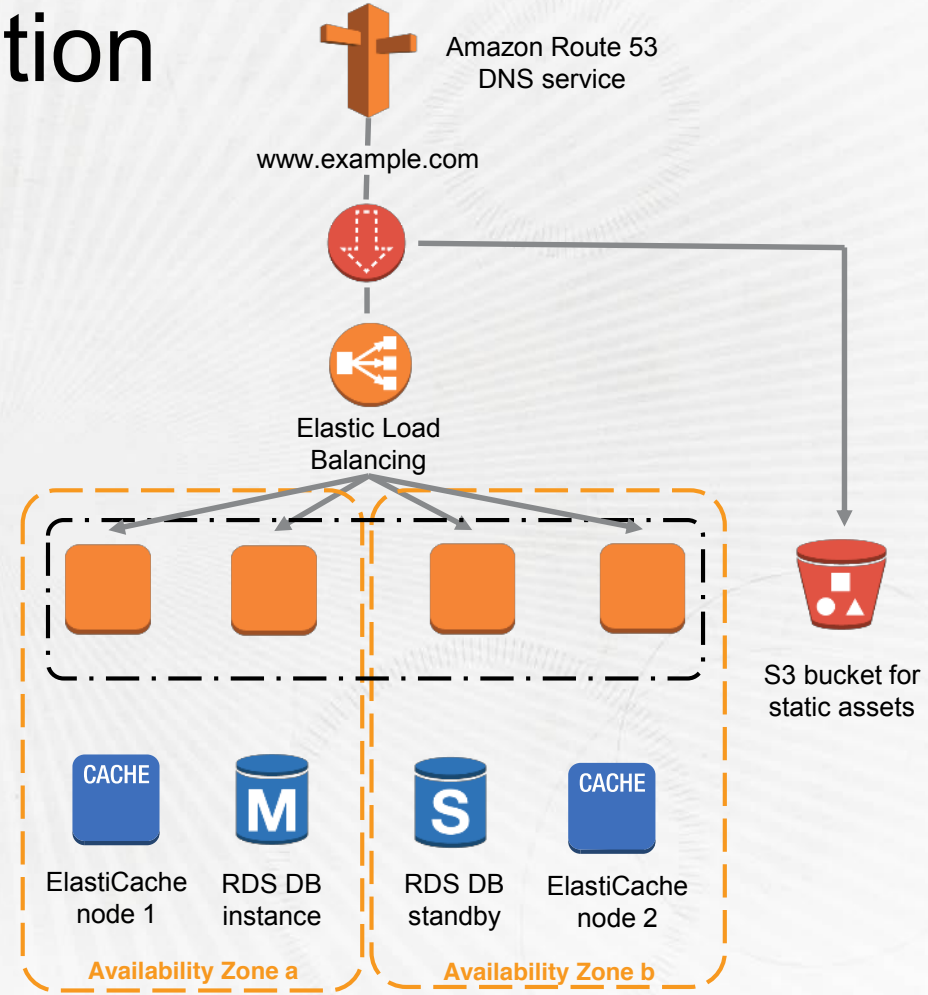
September 15th, 2015 | AWS Pop Up Loft, London

What you will learn in this session

- Deploying an application with AWS Elastic Beanstalk
- Updating with new versions of your app
- Choosing among the AWS services that can help you run applications more easily

Deploying an application

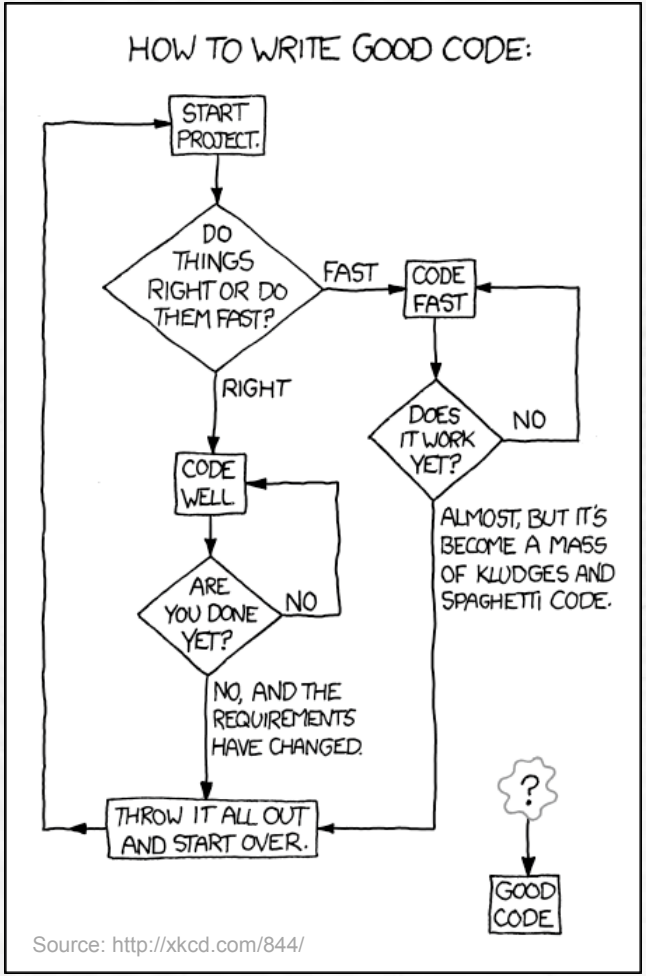
- Web server(s)
- Auto Scaling
- Load balancing
- Database
- Caching
- Security
- Install application code
  - on multiple hosts
  - on multiple environments



It’s not just deployments…

- How do I scale my environment?
- What is i-dc4297f2 used for?
- How do I know if my application is healthy?
- How can I monitor performance?
- Where do I get logs?
- How do I control (or block) SSH access?

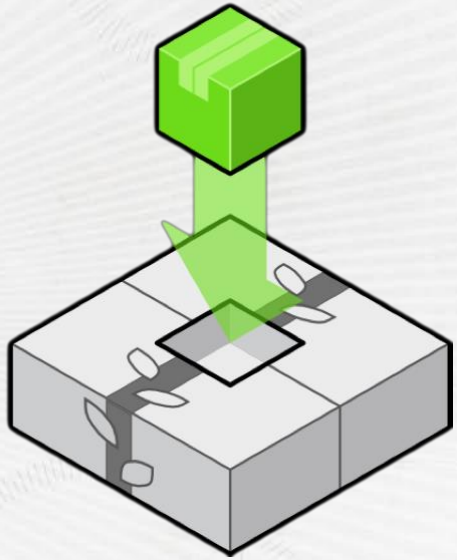
You need to deliver resilient applications with less work



Source: http://xkcd.com/844/

AWS Elastic Beanstalk (EB)

- Easily deploy, monitor, and scale three-tier applications
- Infrastructure provisioned and managed by EB
  - you maintain complete control.
- Preconfigured application containers
  - easily customizable.
- Support for these platforms:



Benefits of Elastic Beanstalk

Focus on your application

Focus on what makes your business unique

Focus on innovation, not undifferentiated heavy lifting

Spend developer time in the right place

Automate as much as you can

There’s no additional charge for Elastic Beanstalk

Elastic Beanstalk object model

Application

Environments

- Infrastructure resources (such as EC2 instances, ELB load balancers, and Auto Scaling groups)
- Runs a single application version at a time for better scalability
- An application can have many environments (such as staging and production)

Application versions

- Application code
- Stored in Amazon S3
- An application can have many application versions (easy to rollback to previous versions)

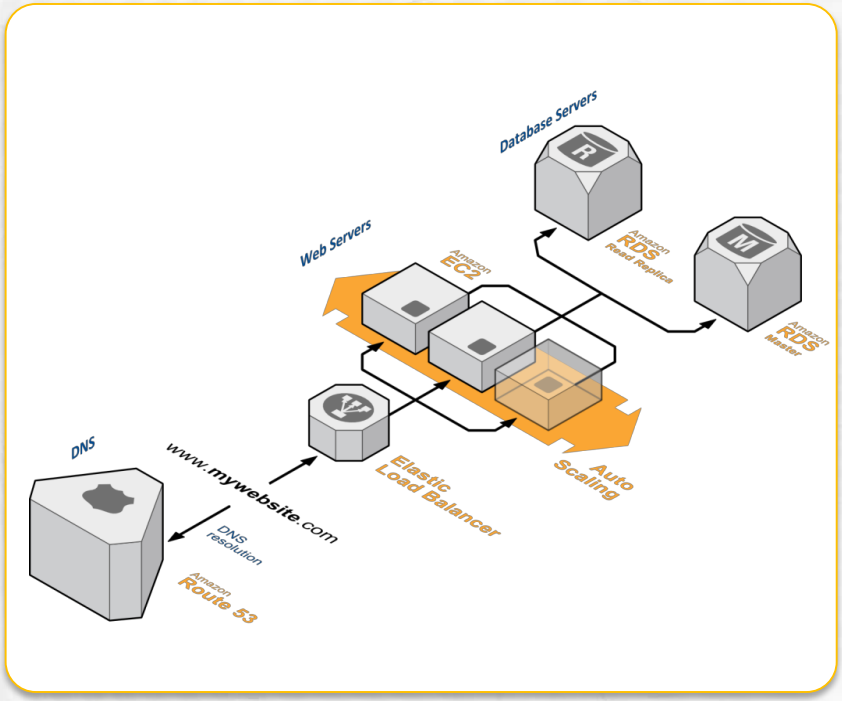
Saved configurations

- Configuration that defines how an environment and its resources behave
- Can be used to launch new environments quickly or roll-back configuration
- An application can have many saved configurations

Elastic Beanstalk environment

- Two types:
  - Single instance
  - Load balancing, auto scaling
- Two tiers (web server and worker)
- Elastic Beanstalk provisions necessary infrastructure resources such as load balancers, auto-scaling groups, security groups, and databases (optional)
- Configures Amazon Route 53 and gives you a unique domain name

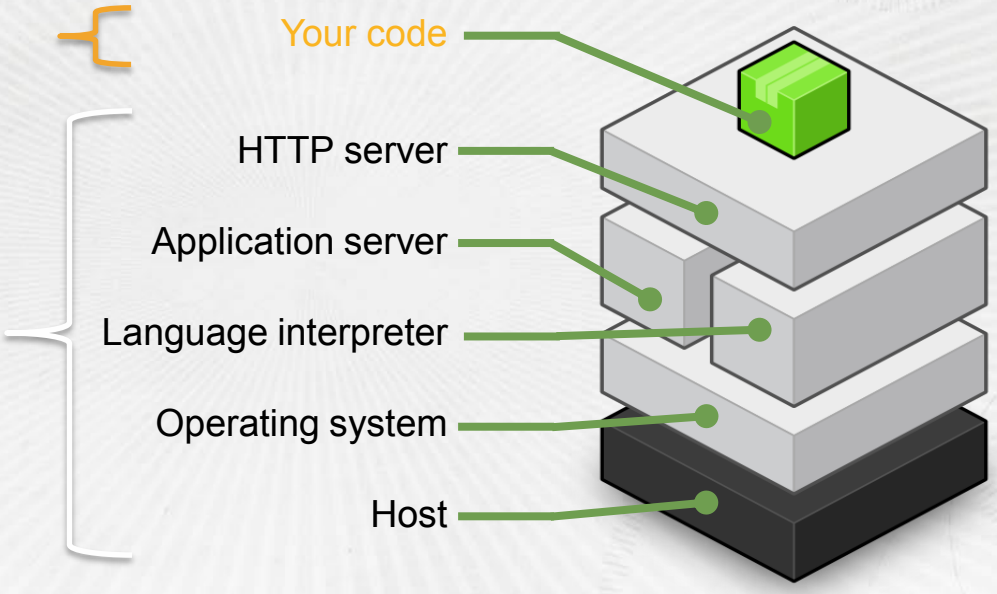
  (For example: yourapp.elasticbeanstalk.com)



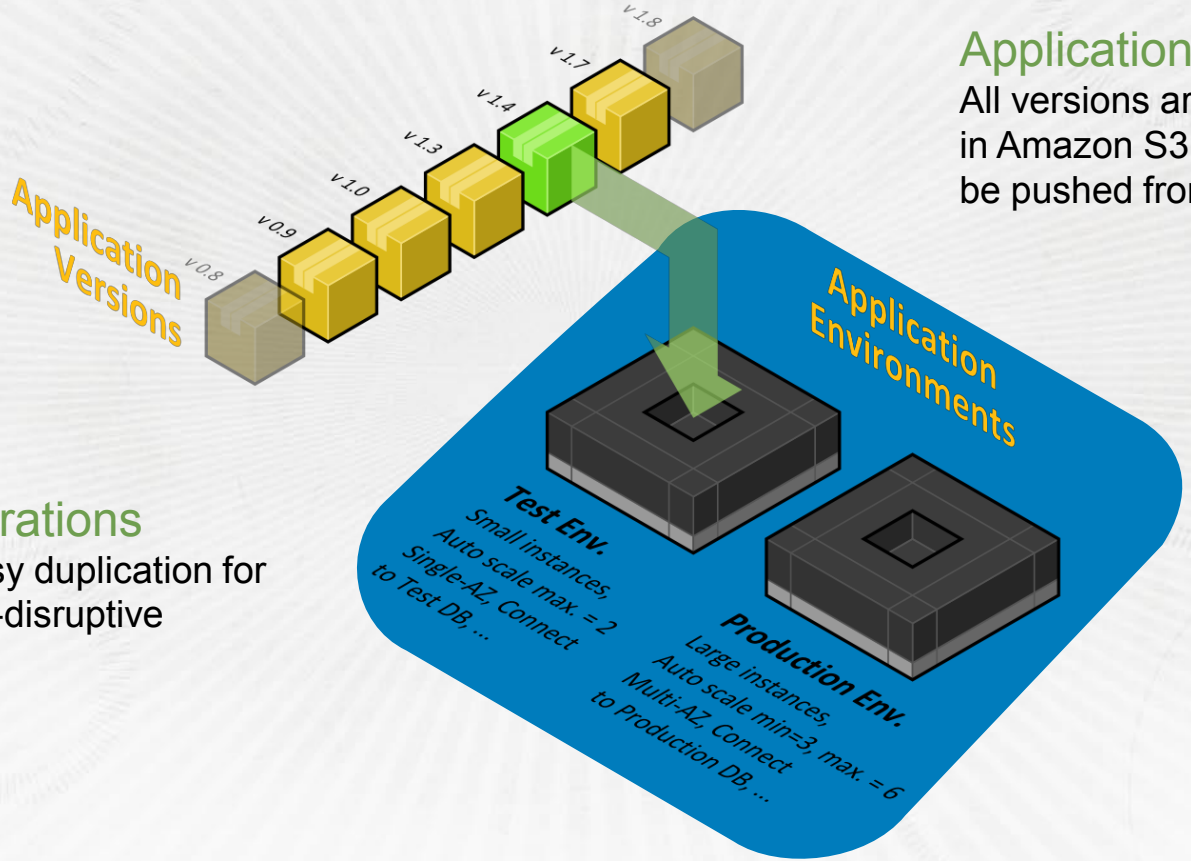
On-instance configuration

Focus on building your application

- Elastic Beanstalk configures each EC2 instance in your environment with the components necessary to run applications for the selected platform
- No more worrying about logging into instances to install and configure your application stack



Application versions and saved configurations



Application versions

All versions are stored durably in Amazon S3. Code can also be pushed from a Git repository

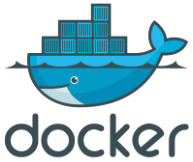
Saved configurations

Save these for easy duplication for A/B testing or non-disruptive deployments

Elastic Beanstalk supports…

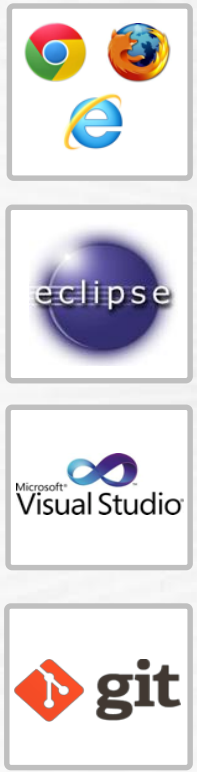
Java	PHP	Python	Ruby	.NET	Node.js
					

Docker

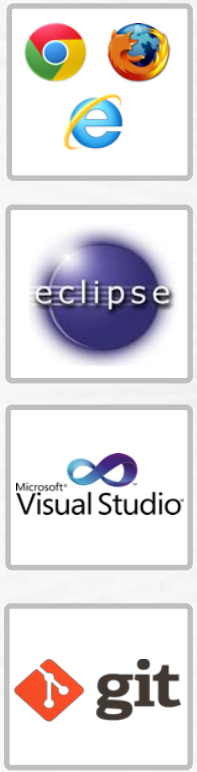


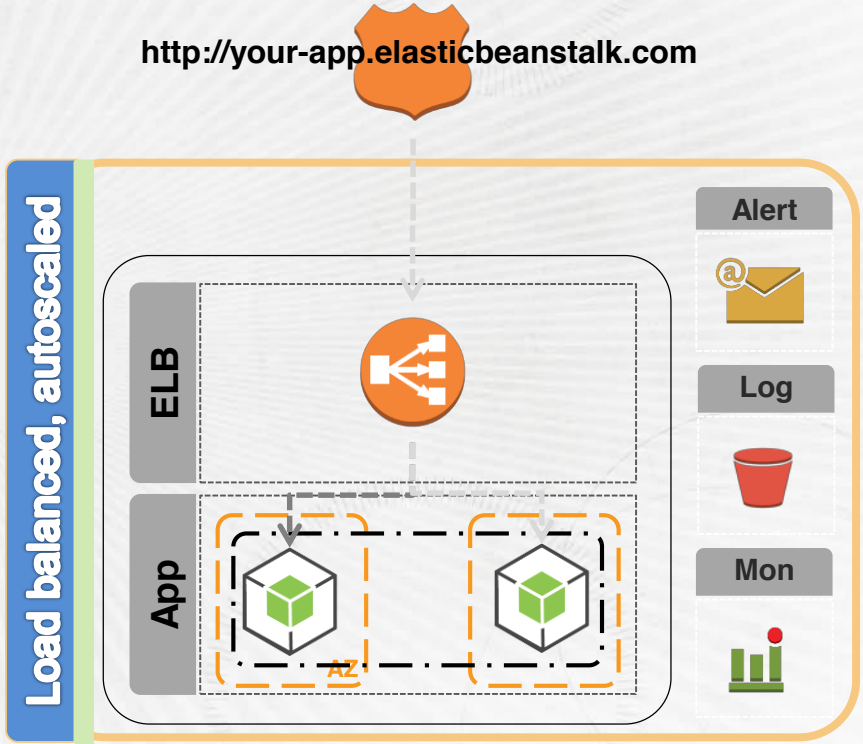
Deploy with tools you know

Deploy with tools you know

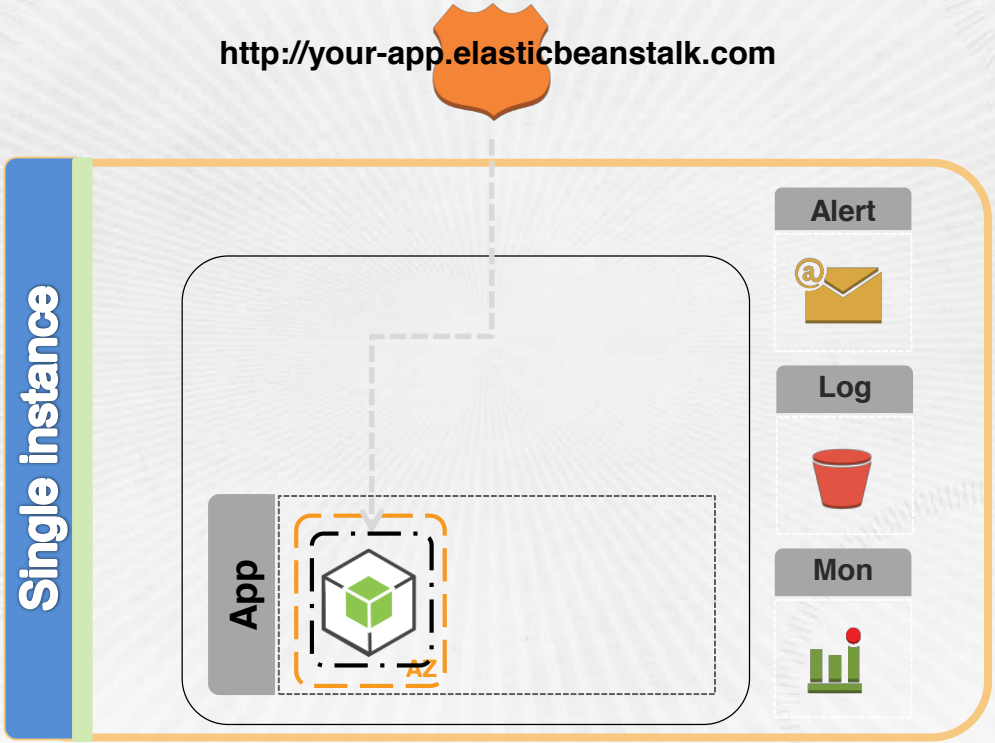


Deploy with tools you know

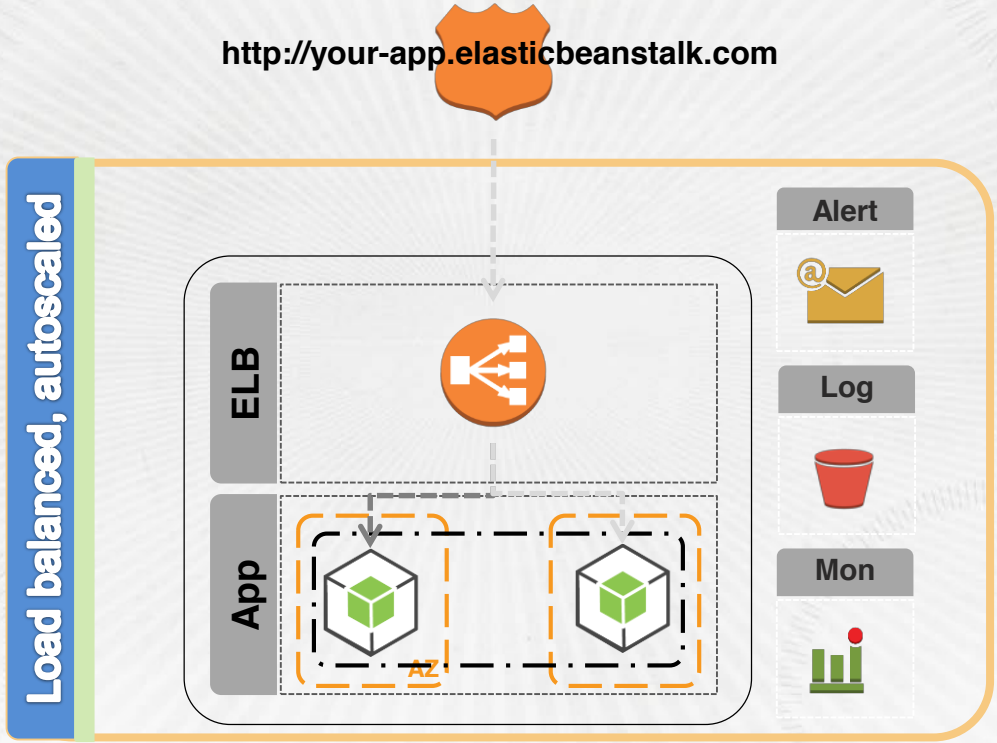




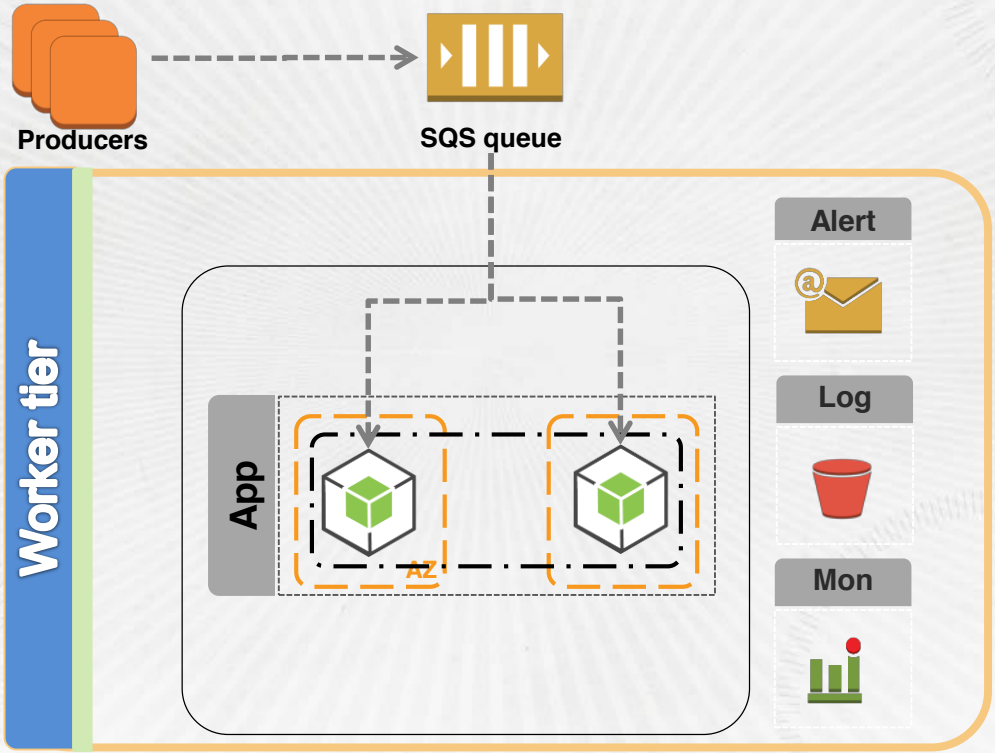
Deploy your app for test



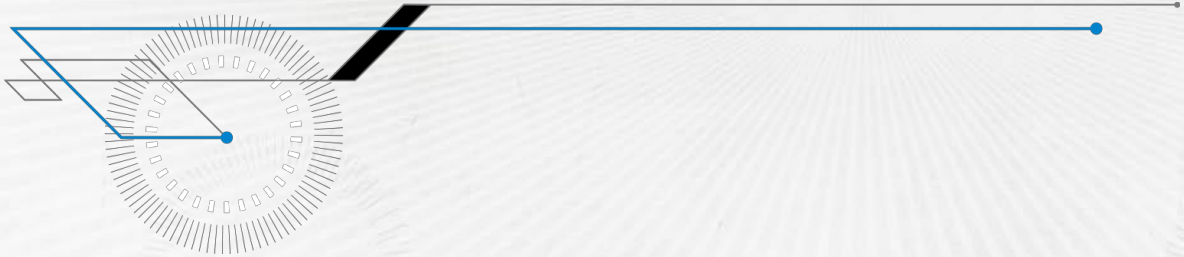
Deploy your app for scale



Deploy your background processing app



Developer workflow

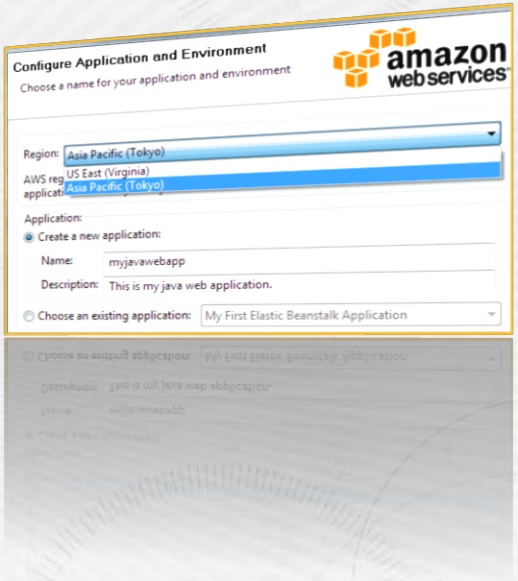


Deployment options

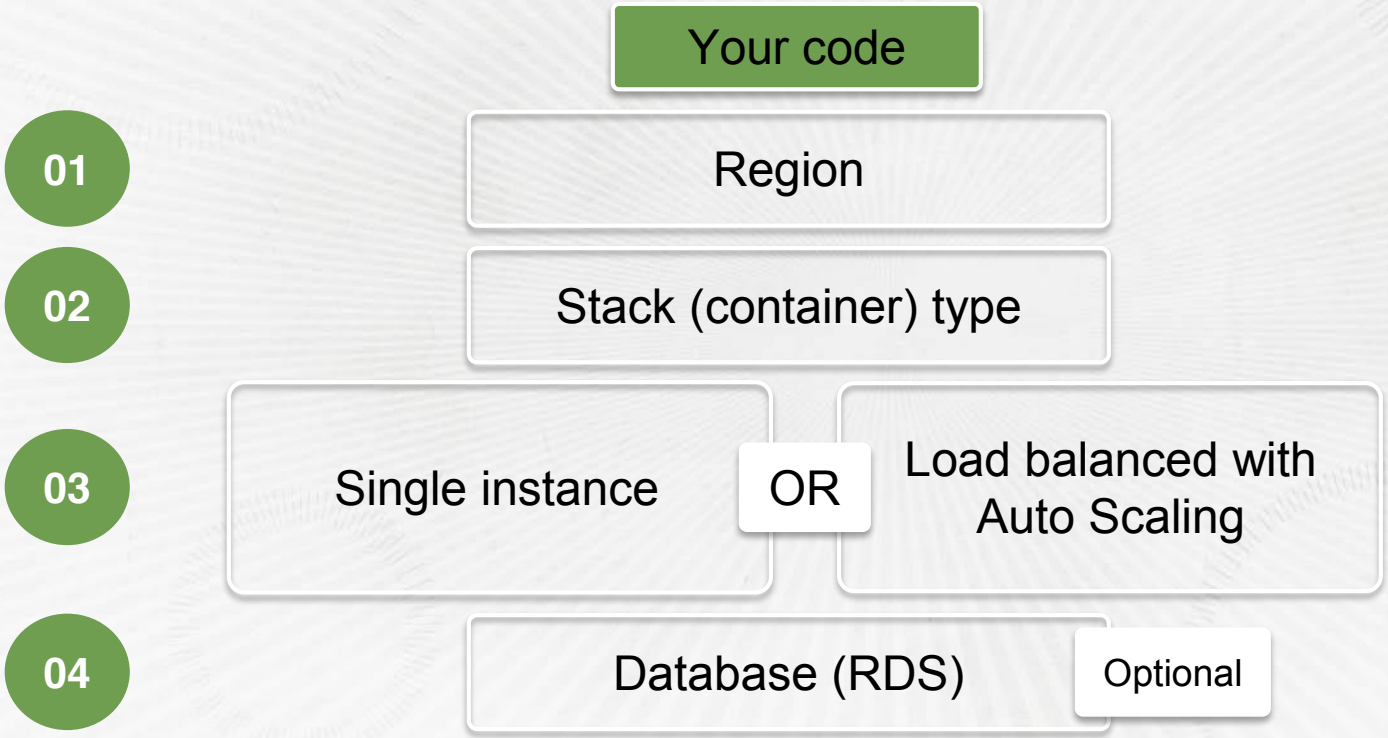
1. Via the AWS Management Console
2. Via Git / EB CLI

```
$ git aws.push
```

3. Via the AWS Toolkit for Eclipse and the Visual Studio IDE



Deployment configuration



Example: CLI workflow

Prerequisites:

- AWS account – your access and secret keys
- EB CLI
  - Linux / Unix / Mac: Python 2.7 or 3.0
  - Windows PowerShell 2.0
- A credential file
- Git 1.66 or later (optional)

Example: CLI workflow

Initial app deployment:

01 Initialize your Git repository

```
$ git init .
```

02 Create your Elastic Beanstalk app

```
$ eb init
```

03 *Follow the prompts to configure the environment*

04 Add your code

```
$ git add .
```

05 Commit

```
$ git commit -m “v1.0”
```

06 Create the resources and launch the application

```
$ eb create
```

Example: CLI workflow

Update your app:

01 Update your code

02 Push the new code

```
$ git add .
$ git commit -m “v2.0”
$ eb deploy
```

03 Monitor the deployment progress

```
$ eb status
```

Example: Deploy Docker container to EB

- Three ways:
  - Dockerfile *(image built on instance).*
  - Dockerrun.aws.json *(manifest file that describes how to run the Docker image).*
  - Application archive *(should include Dockerfile or Dockerrun.aws.json file).*
- Benefits:
  - Enables high-fidelity deployments.
  - You own the runtime. You can use any language or framework, even those not currently supported by Elastic Beanstalk (such as Go, Scala, and Clojure).

Dockerfile

```
FROM dockerfile/nginx

#Add custom index.html
ADD index.html /usr/share/nginx/html/
```

Dockerrun.aws.json

```
{
    "Image":{
        "Name" : "example/wordpress"
    },
    "Ports" : [
        {
            "ContainerPort" : "80"
        }
    ]
}
```

Example: Deploy Docker container to EB

Using the EB command line tool:

01 Initialize your Git repository

```
$ git init .
```

02 Create your Elastic Beanstalk app

```
$ eb init
```

03 *Follow the prompts to configure the environment and copy Dockerfile*

04 Add your code

```
$ git add Dockerfile
```

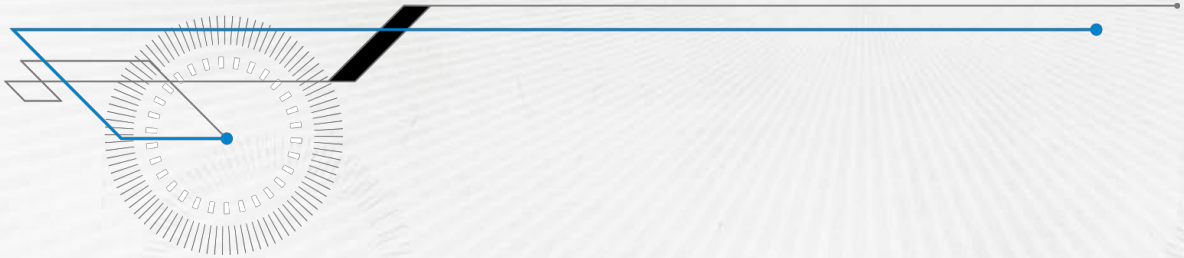
05 Commit

```
$ git commit -am “v1.0”
```

06 Create the resources and launch the application

```
$ eb create
```

Advanced Topics



Zero-downtime deployments

Rolling Deployments

1. EB detaches batch from ELB, Connection draining, Health Checks
2. EB reattaches batch to ELB and moves to next group of instances

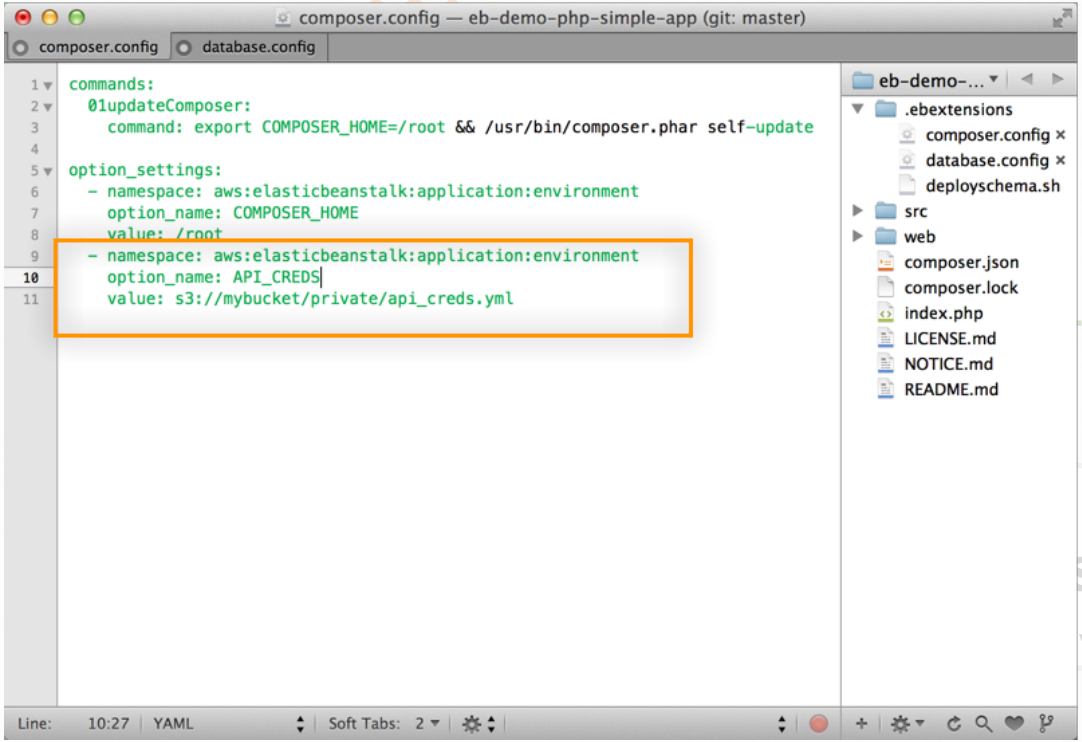
Swap URLs

1. Create a new environment for an existing application
2. Deploy your updated application code to the new environment
3. Use the “Swap URLs” feature to transition users to the new production environment

Is your app multi-environment friendly?

- There will be differences between your application in DEV, TEST, PROD etc
  - DB hostname
  - Logs verbosity…
- Environment specific parameters
  - Should not be hardcoded in your application files
  - Elastic Beanstalk allows you to define environmental variables
  - Defined via the console or in ebextensions

Store sensitive config, safely



Python

```
import os
some_var=os.environ.get('API_CREDS')
```

Java

```
String some_var =
     System.getProperty('API_CREDS')
```

C#

```
NameValueCollection appConfig =
ConfigurationManager.AppSettings;

string param = appConfig["API_CREDS"];
```

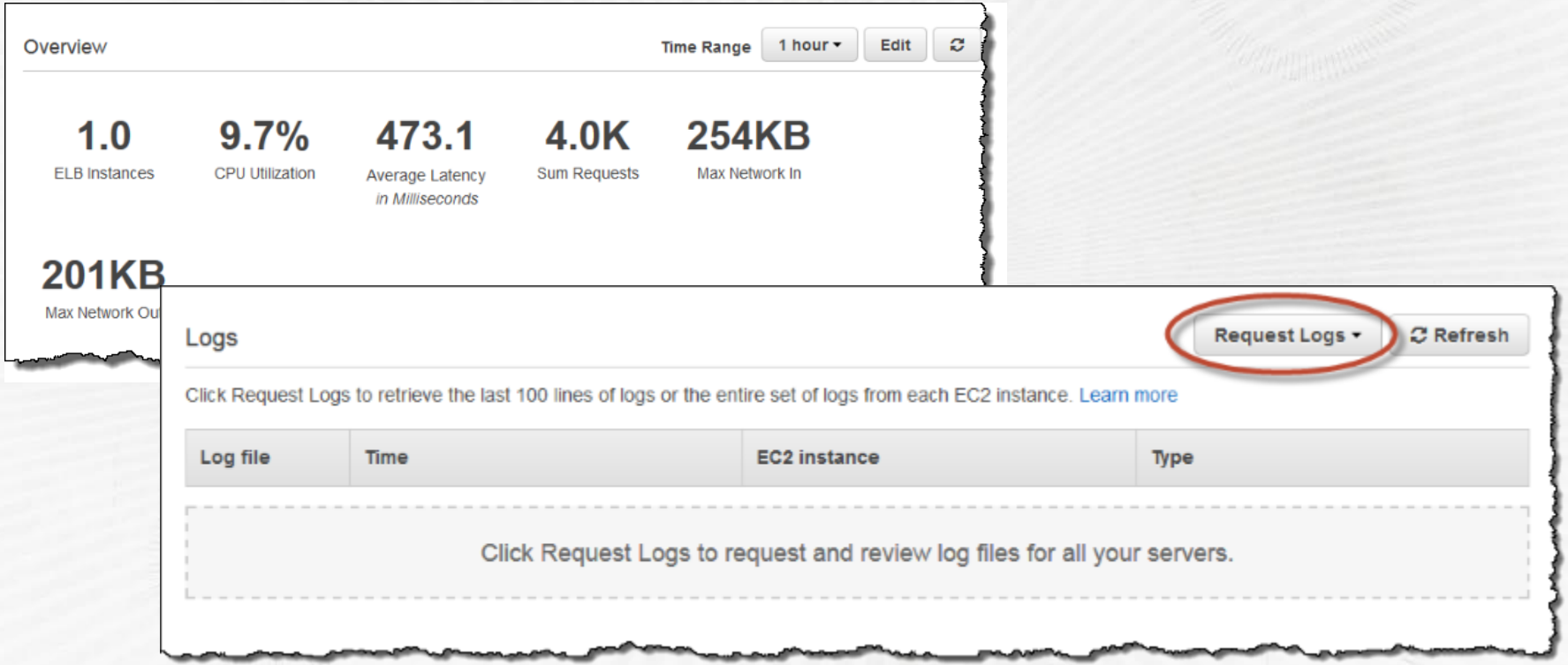
Customize application containers

Add custom software to your environment using ebextensions

```
packages:
  yum:
    newrelic-sysmond: []
  rpm:
    newrelic: http://yum.newrelic.com/pub/newrelic/el5/i386/newrelic-repo-5-3.noarch.rpm

commands:
  0_newrelic_command:
    command: "touch /tmp/$(date '+%F.%T.%N').newrelic_command_0"
  1_configure_new_relic_key:
    command: nrsysmond-config --set license_key=<Your key here>
  1a_newrelic_command:
    command: "touch /tmp/$(date '+%F.%T.%N').newrelic_command_1a"
  2_start_new_relic:
    command: "/etc/init.d/newrelic-sysmond start"
  2a_newrelic_command:
    command: "touch /tmp/$(date '+%F.%T.%N').newrelic_command_2a"
```

Monitoring, Alarms & Logs



Enhanced Health Reporting and Monitoring

- Health agent included in the AMI
- Monitors web server logs and system metrics
- Additional system metrics
- Reports metrics to Elastic Beanstalk every 10 sec
- Optionally reported to CloudWatch every 60 sec

Iterate on application architecture

Add additional resources to your environments using ebextensions

Add other components such as:

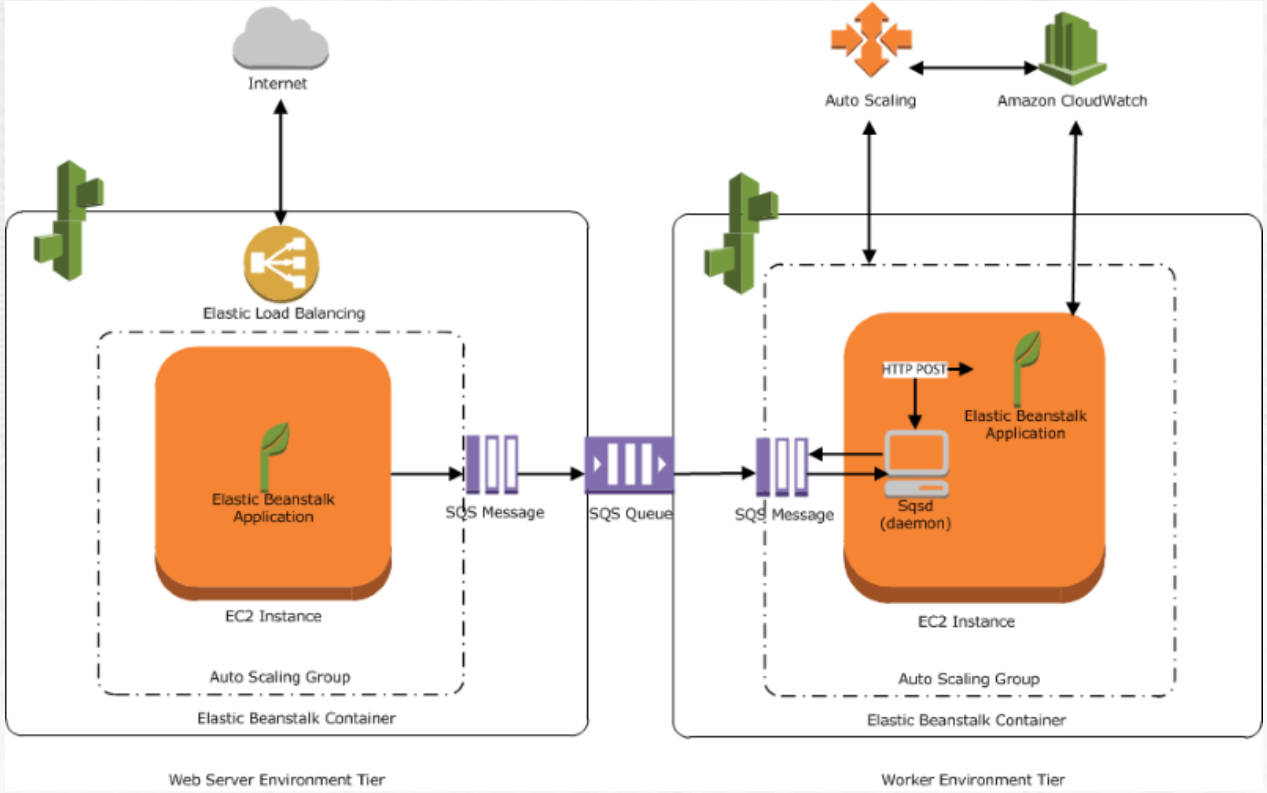
- In-memory caching (Amazon ElastiCache Redis and Memcached)
- Amazon SQS
- Amazon CloudFront

```
Resources:
  MyElastiCache:
    Type: AWS::ElastiCache::CacheCluster
    Properties:
      CacheNodeType:
         Fn::GetOptionSetting:
             OptionName : CacheNodeType
             DefaultValue: cache.m1.small
      NumCacheNodes:
           Fn::GetOptionSetting:
             OptionName : NumCacheNodes
             DefaultValue: 1
      Engine:
           Fn::GetOptionSetting:
             OptionName : Engine
             DefaultValue: memcached
```

Or use CloudFormation => assets independent of environment life time.

Worker Environments

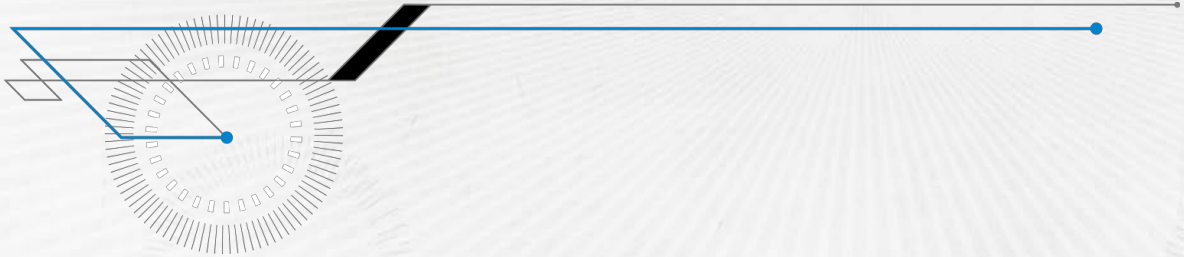
- A daemon on each Amazon EC2 instance processes Amazon SQS messages
- Pulls data off the Amazon SQS queue, inserts it into the message body of an HTTP POST request
- Sends it to a user-configurable URL path on the local host.



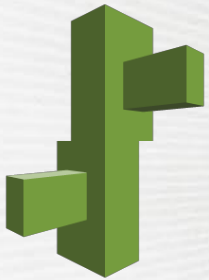
IAM integration

- Service Role vs Instance profile
- Policies
  - Full access for Admins
  - All bar specific actions (create app, create environment) for Devs
  - Read-only for Testers
  - Resource specific permissions (e.g. app1 and app2 only)

DEMO TIME



Application container	Application automation	Templated provisioning



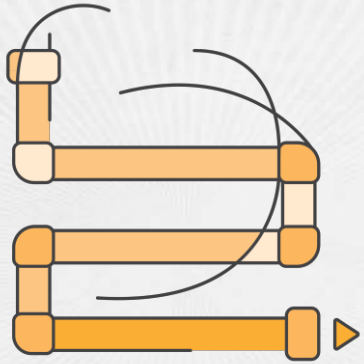
Elastic Beanstalk

OpsWorks

CloudFormation

AWS Code services

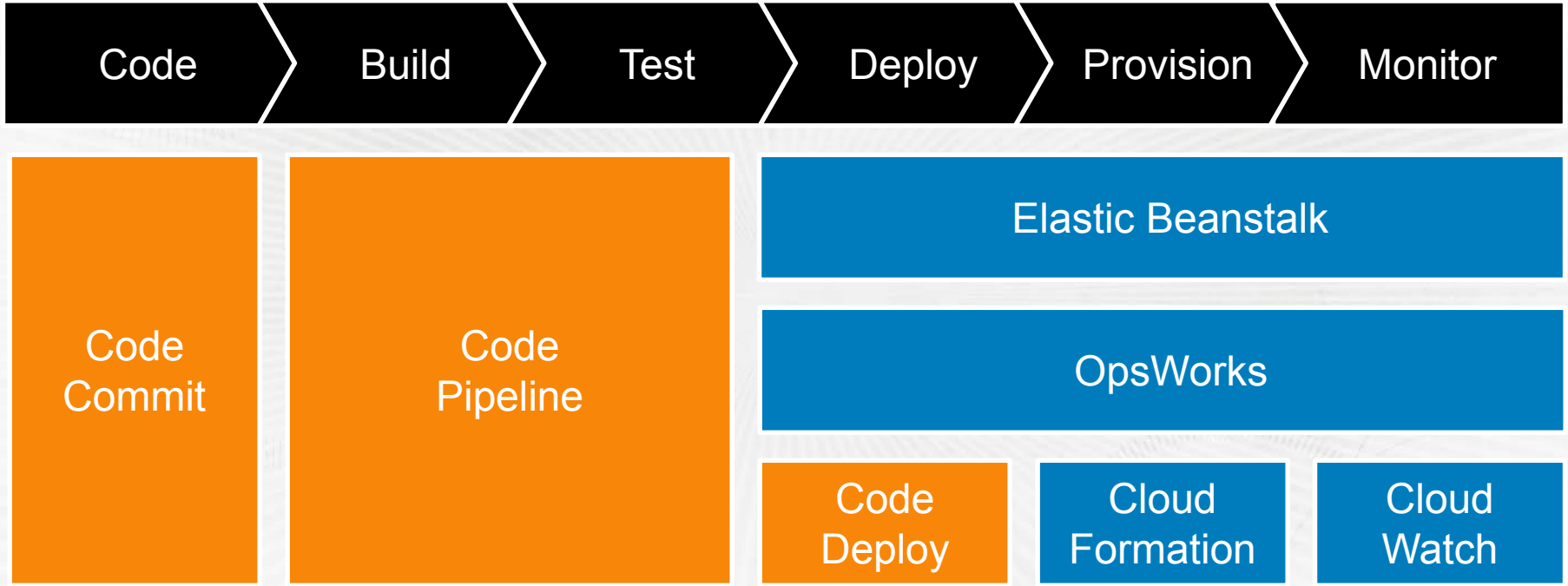
CodeCommit



CodePipeline

CodeDeploy

AWS Code, Deployment & Management Services



What we discussed

- How to get an application running using AWS Elastic Beanstalk
- How to manage multiple environments (QA, Prod)
- How to deploy new versions of your app
- How to choose among the AWS services that can help you run applications more easily

Learn more

Download our free kindle e-book – *Develop, Deploy, and Manage for Scale with AWS Elastic Beanstalk and AWS CloudFormation* (http://amzn.to/10JK5dm)

Read our blog, http://bit.ly/1o6PwZm

Follow us @aws_eb

Thank you!