15-319 / 15-619
Cloud Computing

Recitation 4

Feb 5 , 2019

Administrative - OH & Piazza

- Make use of office hours
  - Make sure that you are able to describe the problem and what you have tried so far
  - Piazza Course Staff
  - Google calendar in ET
  - Google calendar in PT

- Suggestions for using Piazza
  - Contribute questions and answers
  - Read the Piazza Post Guidelines (@6) before asking questions
  - Read Piazza questions & answers carefully to avoid duplicates
  - Don’t ask a public question about a quiz question
  - Try to ask a **public** question if possible

Administrative - Cloud spending

- Suggestion on cloud service usage
  - Monitor AWS expenses regularly
  - Always do the cost calculation before launching services
  - Terminate your instances when not in use
  - Stopped instances have EBS costs ($0.1/GB-Month)
  - Make sure spot instances are tagged right after launch

Important Notice

- **DON’T EVER EXPOSE YOUR AWS CREDENTIALS!**
  - Github
  - Bitbucket
  - Anywhere public…
- **DON’T EVER EXPOSE YOUR GCP CREDENTIALS!**
- **DON’T EVER EXPOSE YOUR Azure CREDENTIALS!**
  - ApplicationId, ApplicationKey
  - StorageAccountKey, EndpointUrl

Reflection

- Conceptual content on OLI
  - Modules 3, 4, Quiz 2
- Project theme - **Big data analytics**
  - **Inverted Index:** Implemented an inverted index with MapReduce using TDD
  - **Wiki Data Parallel Processing Analysis:** Use MapReduce to process **36GB** compressed / **128GB** uncompressed wiki data
    - MapReduce application to filter records and calculate aggregate daily pageviews
  - **Data Analytics:** Use Jupyter Notebooks and the pandas library to analyze the data and answer questions

This Week

- **Quiz 3 (OLI Modules 5 & 6)**
  - Due on **Friday**, Feb 8th, 2019, 11:59PM ET
- **Project 2.1**
  - Due on **Sunday**, Feb 10th, 2019, 11:59PM ET
- **Primers released this week**
  - P2.2 - Intro to Containers and Docker
  - P2.2 - Kubernetes and Container Orchestration

Guideline for Project Reflection

- Describe your approach in solving each task in this project
  - Please share your:
    - Approach to solving each task in the project
    - What you would do differently if you could do the project over again
    - Interesting problems that your overcame
  - However, please:
    - Do not share your code or pseudocode
    - Do not share details about your solution

OLI Module 5 - Cloud Management

Cloud Software stack - enables provisioning, monitoring and metering of virtual user “resources” on top of the Cloud Service Provider’s (CSP) infrastructure.

- Cloud middleware
- Provisioning
- Metering
- Orchestration and automation
- Case Study: Openstack - Open-source cloud stack implementation

OLI Module 6 - Cloud Software Deployment Considerations

- Programming the cloud
- Deploying applications on the cloud
- Build fault-tolerant cloud services
- Load balancing
- Scaling resources
- Dealing with tail latency
- Economics for cloud applications

Project 2 Overview

Scaling and Elasticity with

- VMs
- Containers
- Functions

- **2.1 Scaling Virtual Machines**
  - Horizontal scaling in / out using AWS APIs
  - Load balancing, failure detection, and cost management on AWS
  - Infrastructure as Code (Terraform)
- **2.2 Scaling with Containers**
  - Building your own container-based microservices
  - Docker containers
  - Manage multiple Kubernetes Cluster
  - Multi Cloud deployments
- **2.3 Functions as a Service**
  - Develop event driven cloud functions
  - Deploy multiple functions to build a video processing pipeline

Project 2.1 Learning Objectives

- **Design** solutions and invoke cloud APIs to programmatically provision and deprovision cloud resources based on the current load.
- **Explore** the usability and performance of APIs used in AWS.
- **Configure** and deploy an Elastic Load Balancer along with an Auto Scaling Group on AWS.
- **Develop** elasticity policies to maintain the QoS of a web service that also deals with resource failure.
- **Account** for cost as a constraint when provisioning cloud resources and analyze the performance tradeoffs due to budget restrictions.
- **Experience** using cloud orchestration and automation tools such as Terraform.

Overview of Quality of Service (QoS), Latency and Cloud Elasticity

- Load patterns for web services
- Vertical scaling (Scale up/down)
- Horizontal scaling (Scale out/in)
- Load balancers
- Autoscaling groups
- Resource monitoring (CloudWatch)
- Resource orchestration with Terraform

Quality of Service (QoS)

Quantitatively Measure QoS

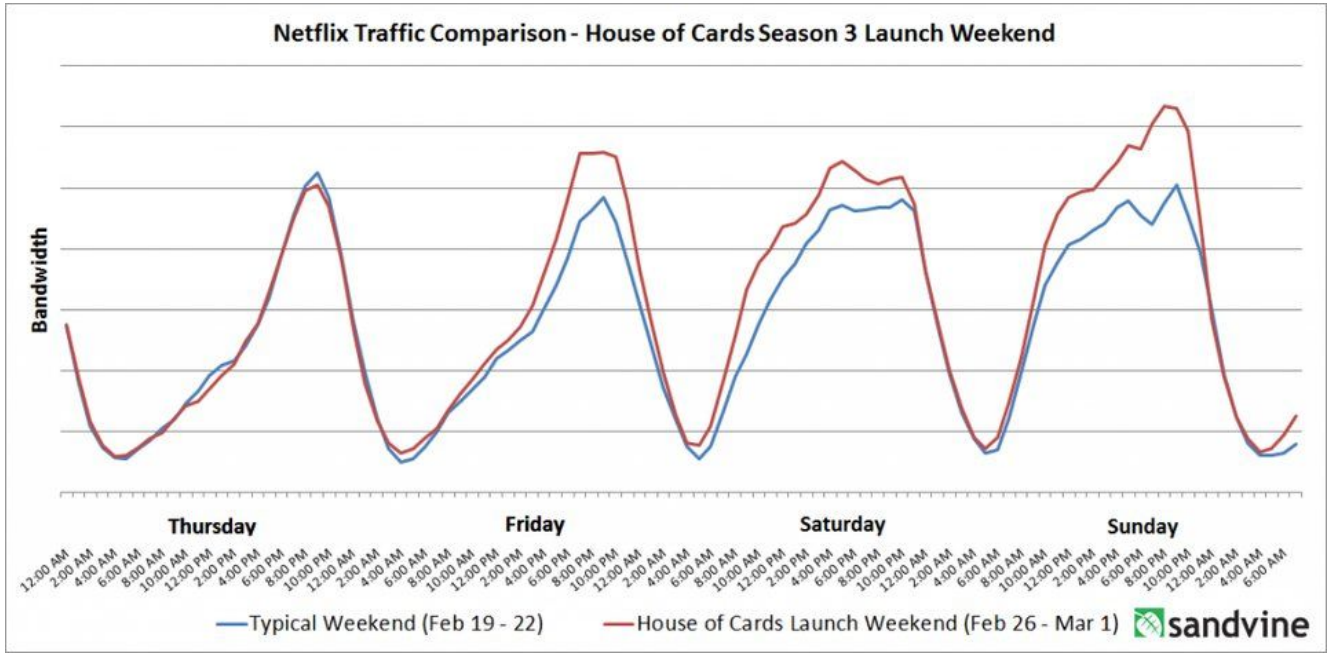
- Performance: Throughput, Latency
  (Very helpful in Projects 2 & Team Project)
- Availability: the probability that a system is operational at a given time *(Projects P2.1 and P2.2)*
- Reliability: the probability that a system will produce a correct output up to a given time *(Project P2.1 and P2.2)*

QoS Matters:

- Amazon found every 100ms of latency cost them 1% in sales (~$1B).

Reality, human patterns...

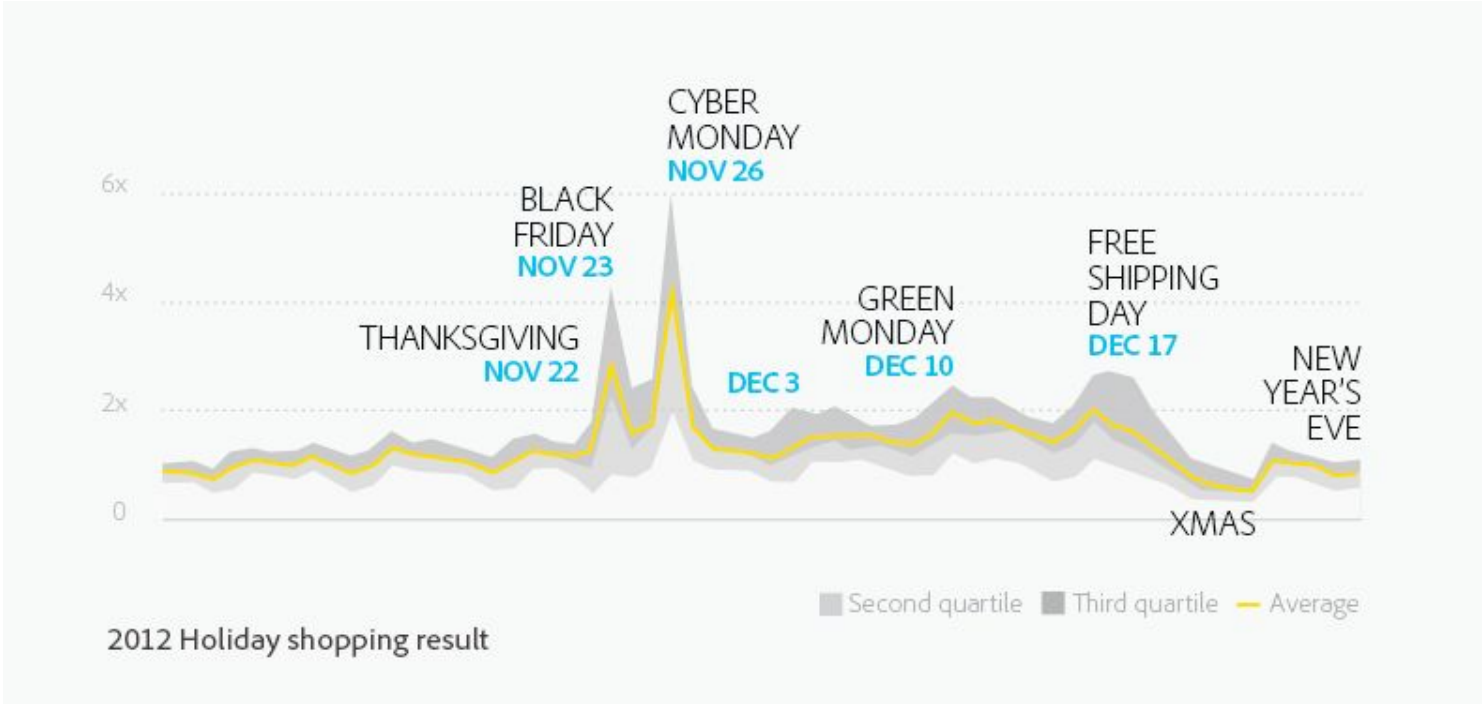
- **Daily**
- **Weekly**
- Monthly
- Yearly
- ...



The Ferenstein Wire

Reality, human patterns...

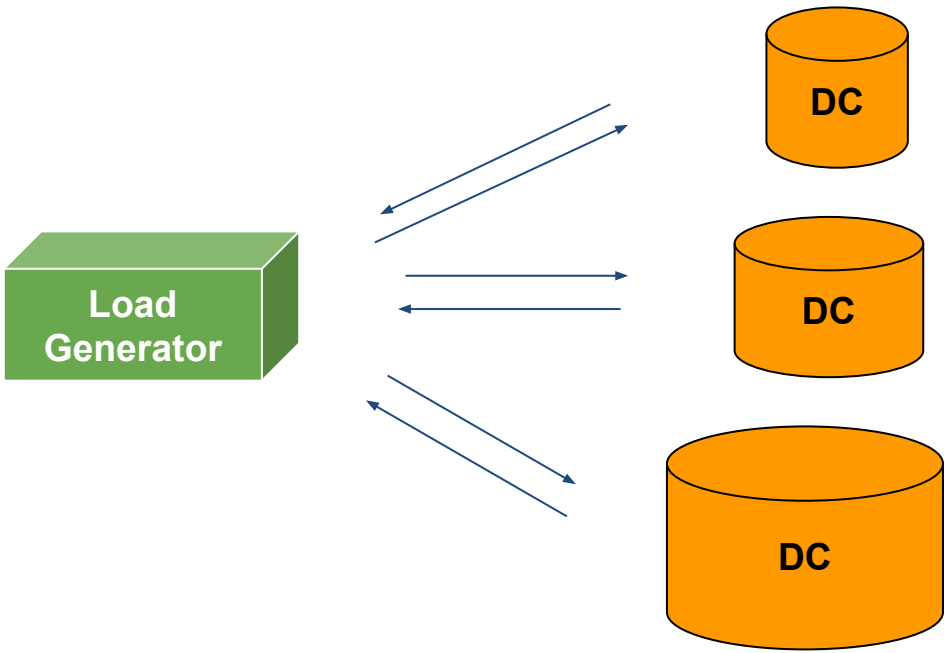
- Daily
- Weekly
- Monthly
- **Yearly**
- ...



sapient.com

Cloud Comes to the Rescue!
Scaling!

P0: Vertical Scaling

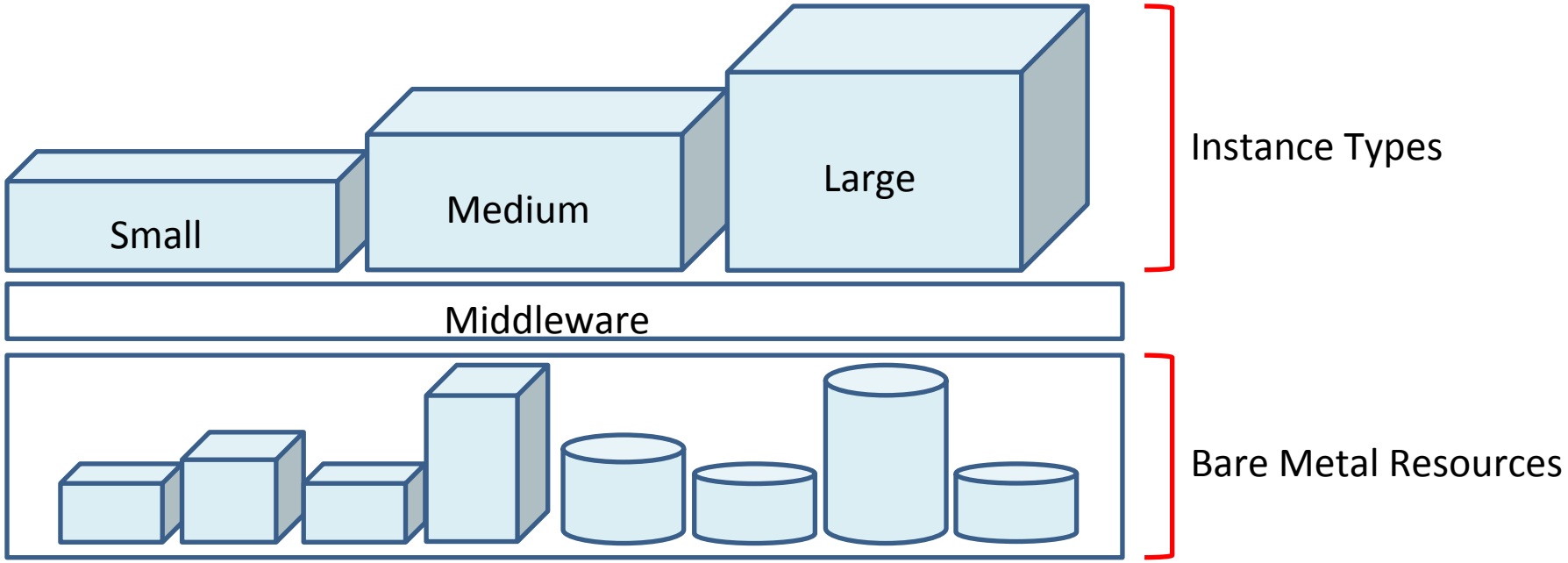


Small Data Center Instance
Serves 100 requests per second

Medium Data Center Instance
Serves 1000 requests per second

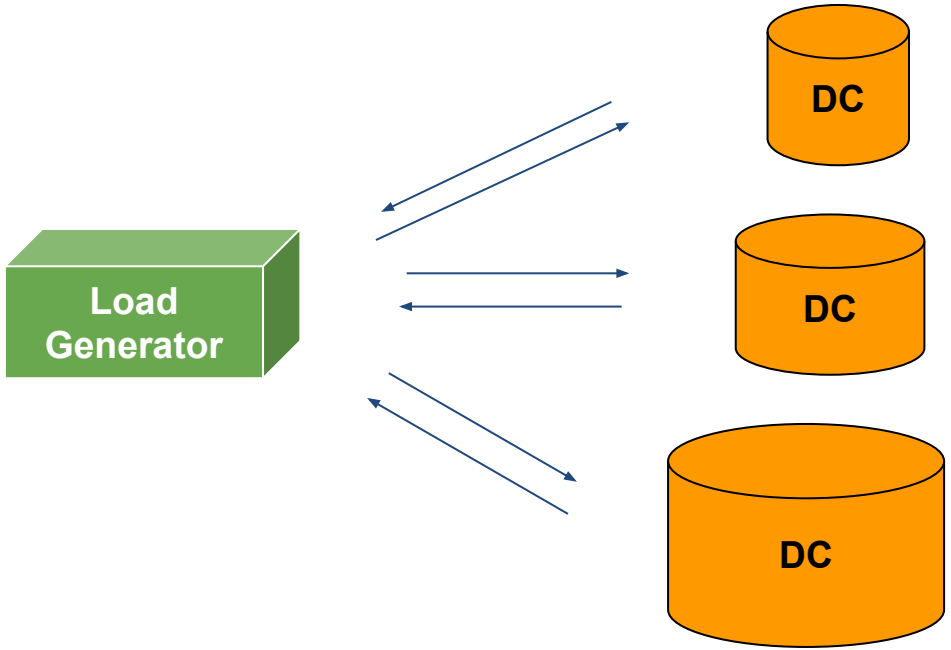
Large Data Center Instance
Serves 10000 requests per second

Resources in Cloud Infrastructure

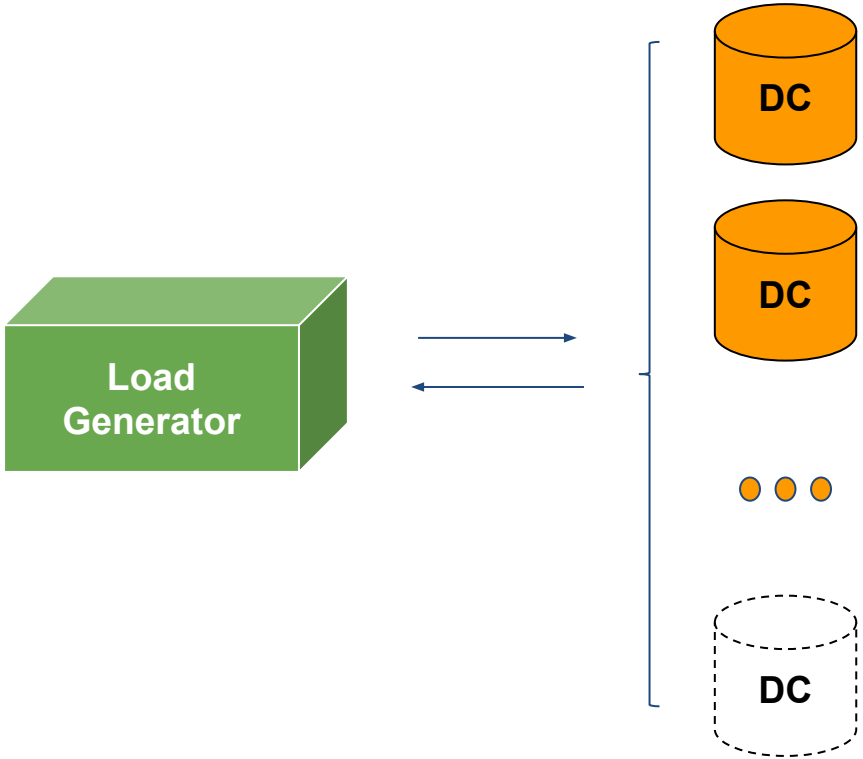


P0: Vertical Scaling Limitation

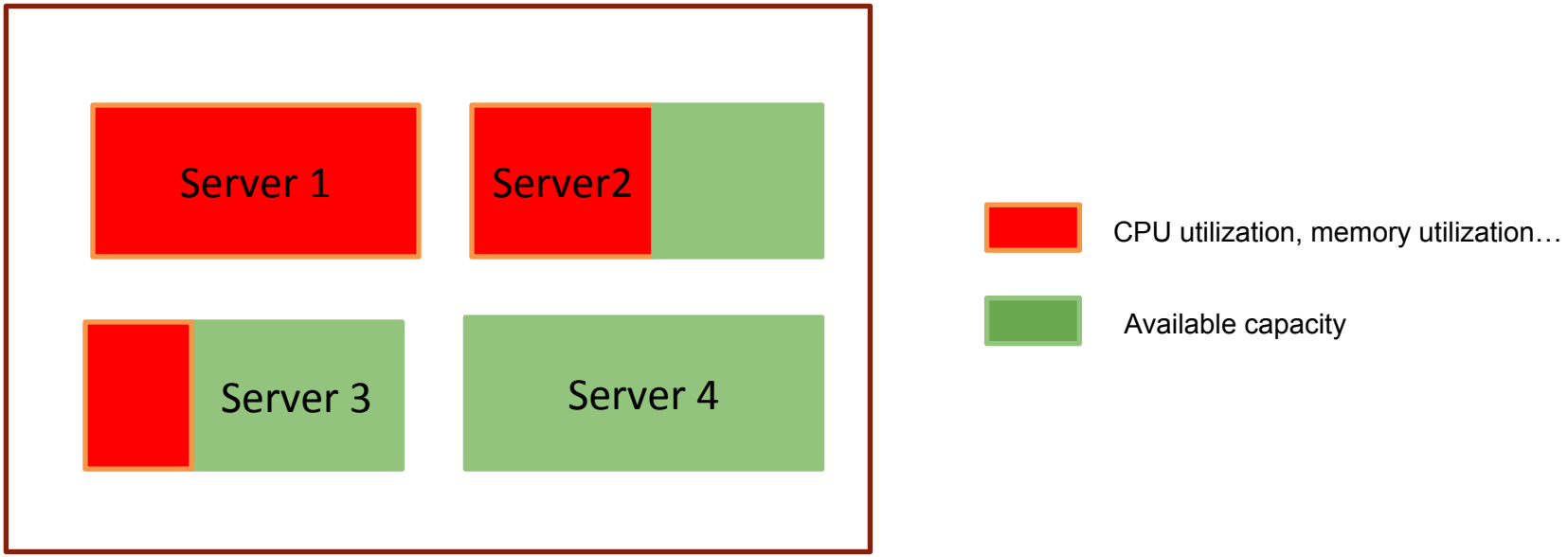
- However, one instance will always have limited resources.
- Reboot/Downtime.



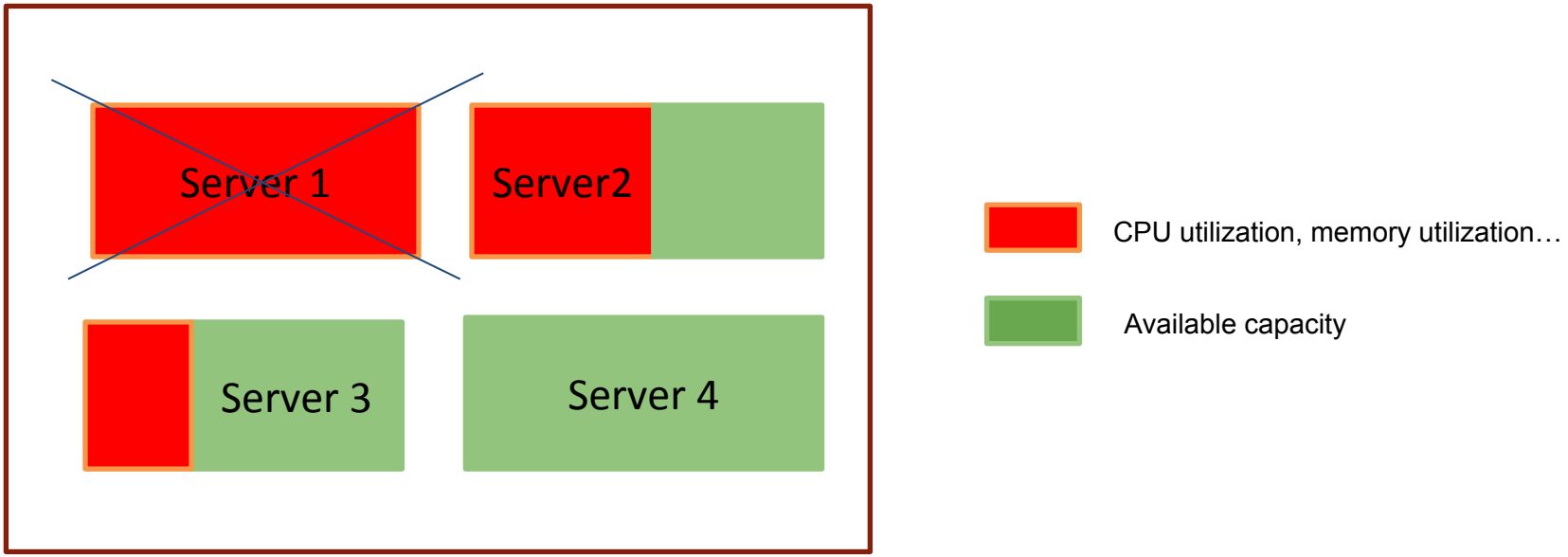
Horizontal Scaling



How do we distribute load?



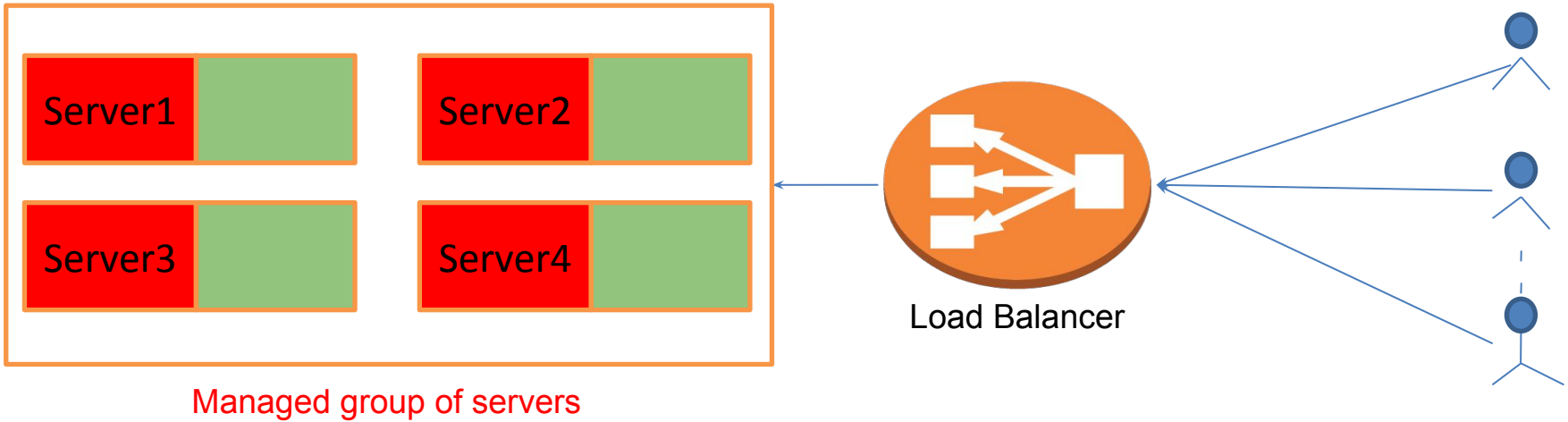
Instance Failure?



What You Need

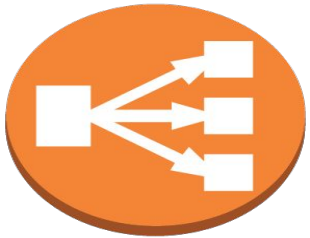
- Make sure that workload is even on each server
- Do not assign load to servers that are down
- Increase/Remove servers according to changing load

How does a cloud service help solve these problems?



Load balancer

- “Evenly” distribute the load
- Simplest distribution strategy
  - Round Robin
- Health Check



Load Balancer

- What if the Load Balancer becomes the bottleneck?
  - Elastic Load Balancer (ELB)
    - Could scale up based on load
  - Elastic, but it still takes time
    - Through the warm-up process

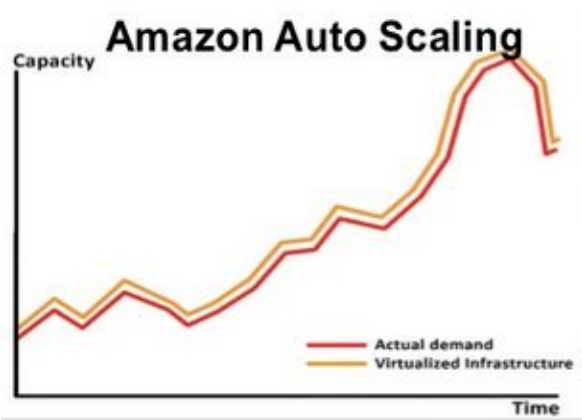
Scaling

Manual Scaling:

- Expensive on manpower
- Low utilization or over provisioning
- Manual control
- Lose customers

Autoscaling:

- Automatically adjust the size based on demand
- Flexible capacity and scaling sets
- Save cost



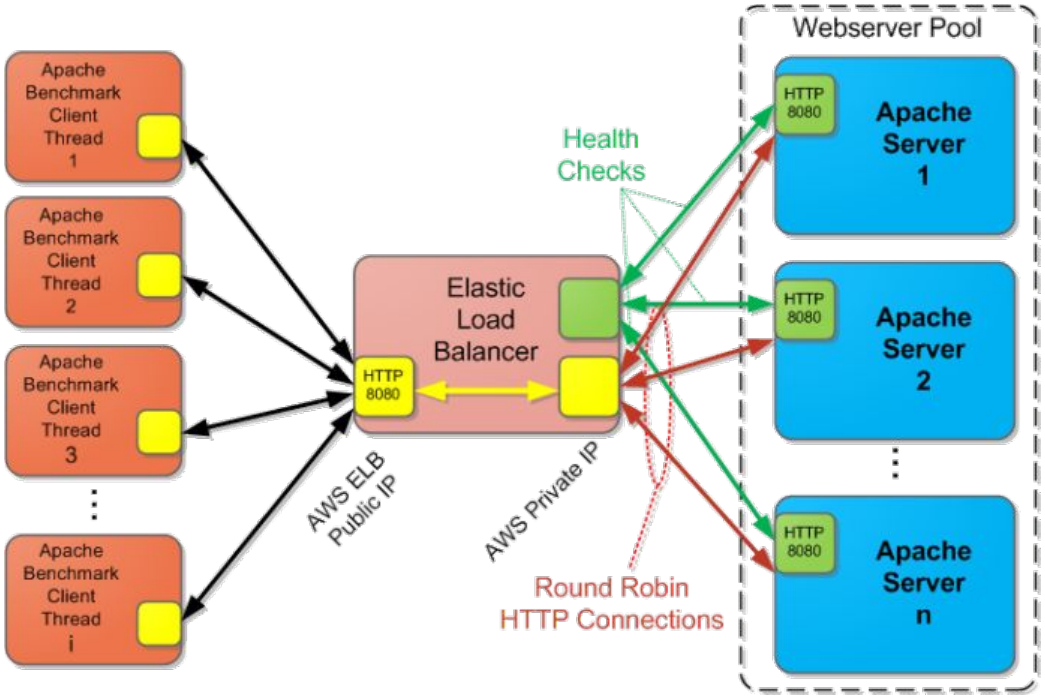
AWS Autoscaling

Auto Scaling on AWS

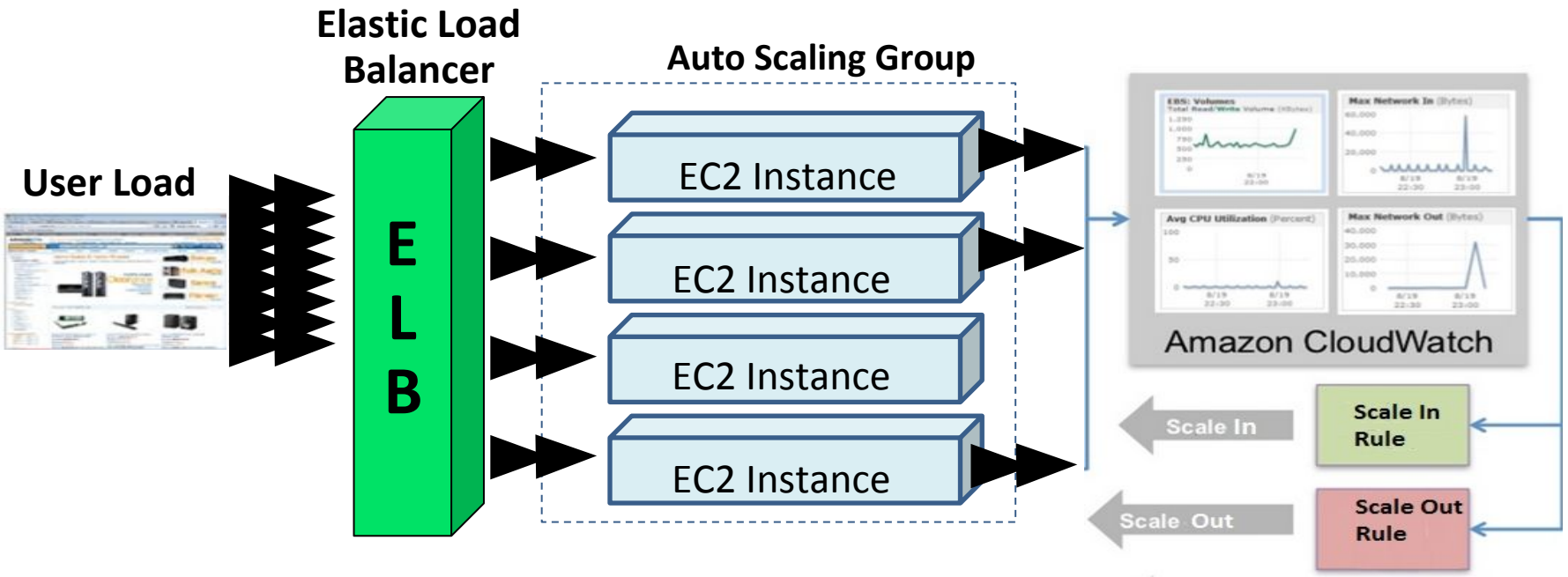
Using the AWS APIs:

- **ELB**
- **Auto Scaling Group**
- **EC2**
- **CloudWatch**
- **Auto Scaling Policy**

You can build a load balanced auto-scaled web service.

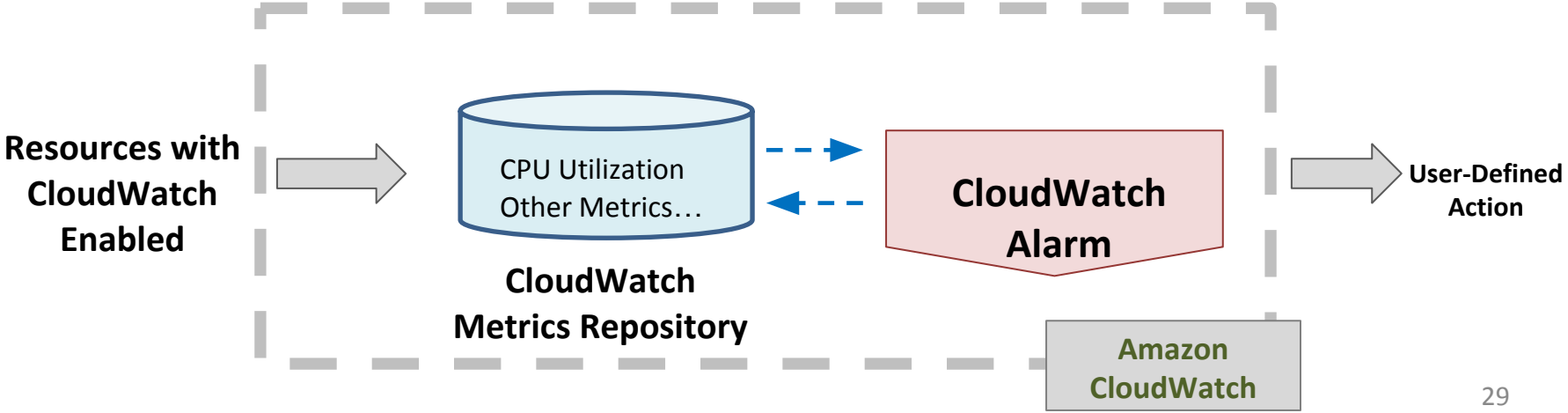


Amazon Auto Scaling Group



Amazon's CloudWatch Alarm

- Monitor CloudWatch metrics for some specified alarm conditions
- Take automated action when the condition is met



Terraform Configuration

- **Providers**
  - A provider is responsible for understanding API interactions and exposing resources.
- **Resources**
  - The resource block defines a resource that exists within the infrastructure.
- **AWS Provider**
  - https://www.terraform.io/docs/providers/aws/index.html

$ **cat main.tf**

```
provider "aws" {
  region     = "us-east-1"
}

resource "aws_instance" "cmucc" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"

  tags {
    Project = "2.1"
  }

  key_name = "my-ssh-key"
}
```

Terraform CLI

- **init**
  - Initializes a working directory containing Terraform configuration files.
- **plan**
  - Creates an execution plan.
- **apply**
  - Apply the changes required to reach the desired state.
- **destroy**
  - Destroy the Terraform-managed infrastructure.

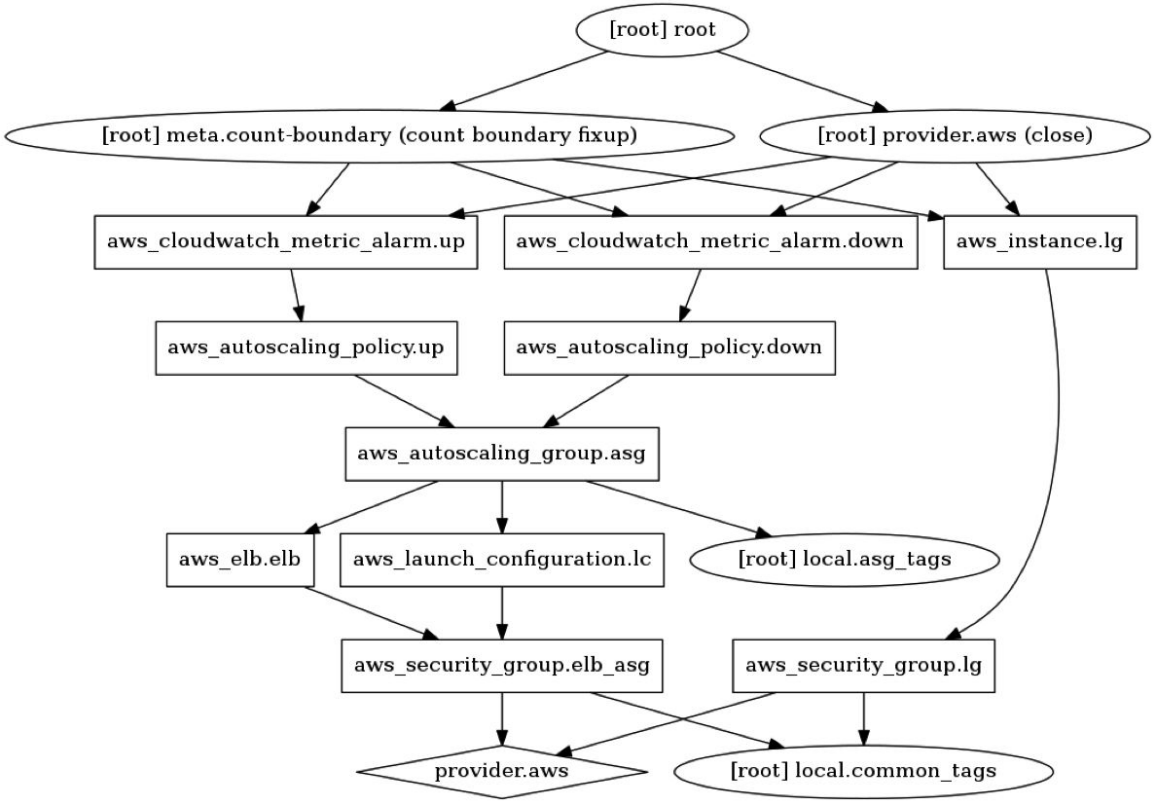
```
$ terraform plan
...
Terraform will perform the following actions:

  + aws_instance.cmucc
      id:                   <computed>
      ami:                  "ami-2757f631"
      ...

Plan: 1 to add, 0 to change, 0 to destroy.

$ terraform apply
...
```

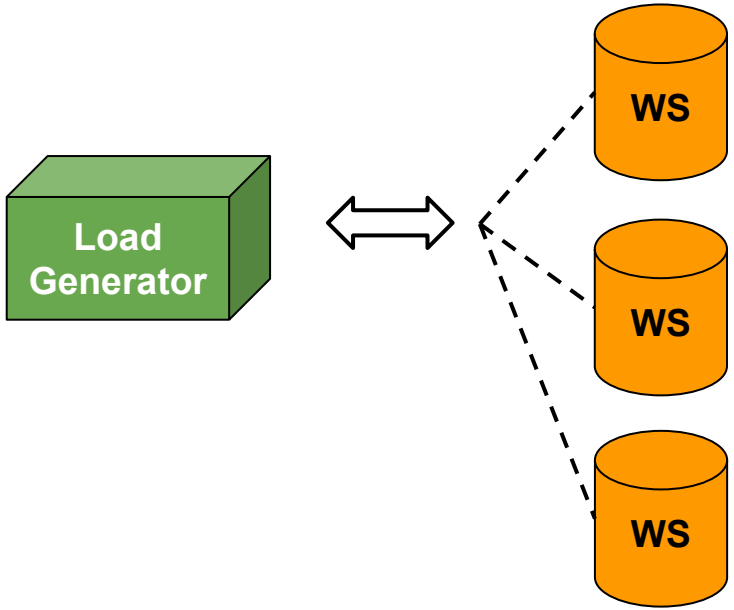
Declarative Infrastructures with Terraform



Do not expect this to exactly match your solution (e.g. the load balancer is supposed to be an Application Load Balancer)!

Project 2.1 Scaling on AWS

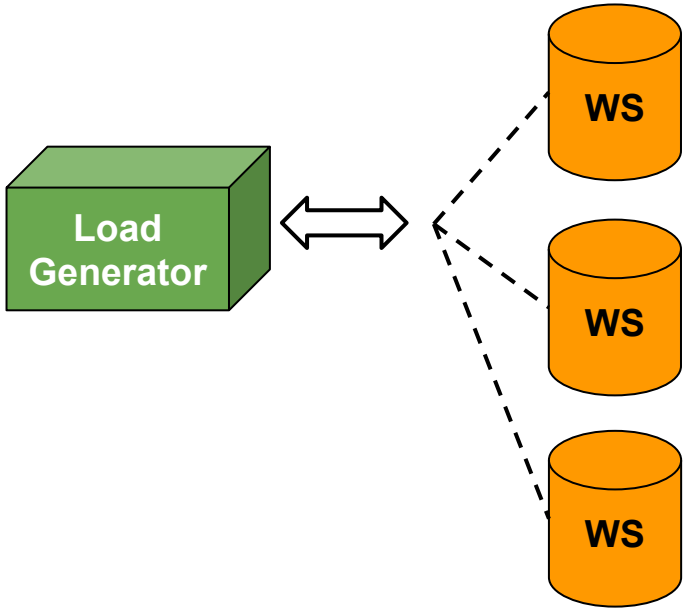
- **Task 1**
  - **AWS Horizontal Scaling**
- Task 2
  - AWS Auto Scaling
- Task 3
  - AWS Auto Scaling with Terraform



Project 2.1 Scaling on AWS

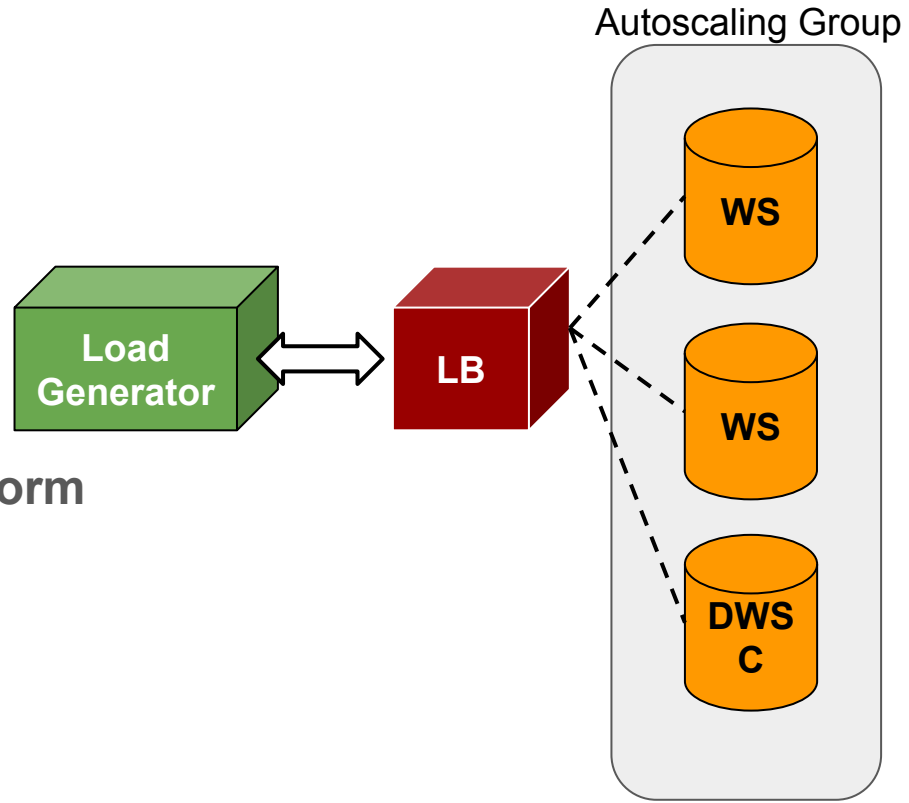
Task 1 - AWS Horizontal Scaling:

- Implement Horizontal Scaling in AWS.
- Write a program that launches the data center instances and ensures that the target total RPS is reached.
- Your program should be fully automated: launch LG->submit password-> Launch DC-> start test-> check log -> add more DC...



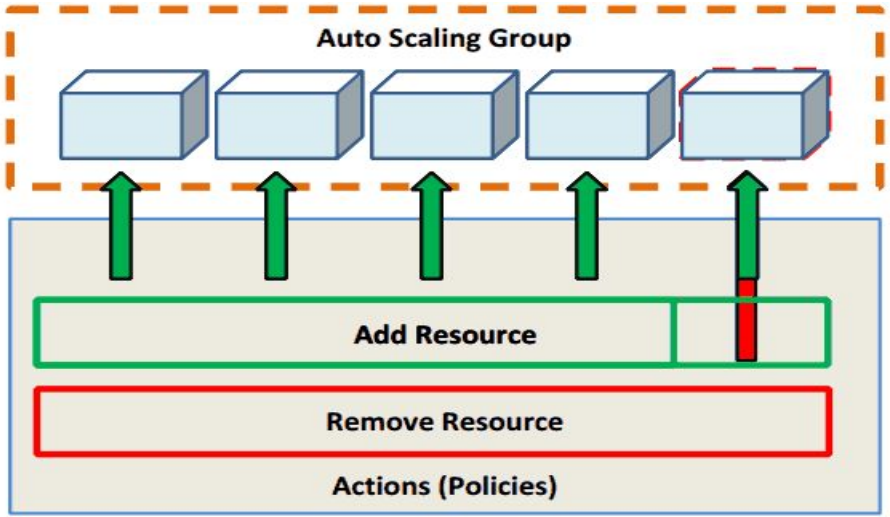
Project 2.1 Scaling on AWS

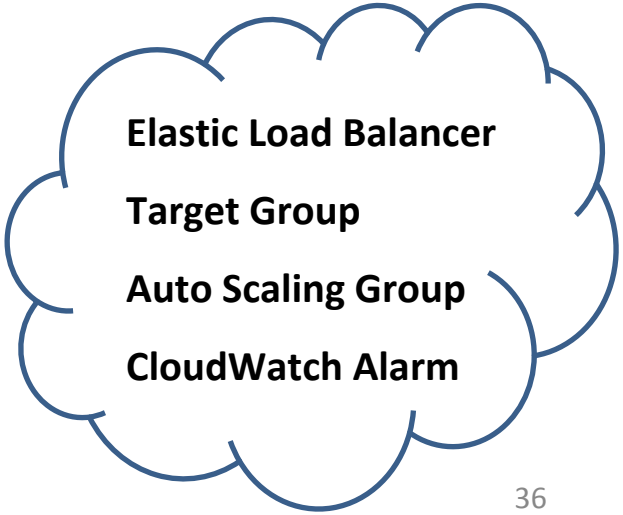
- Task 1
  - AWS Horizontal Scaling
- **Task 2**
  - **AWS Auto Scaling**
- Task 3
  - AWS Auto Scaling with Terraform



P2.1 - Task 2

- Programmatically create an Application Load Balancer (ALB) and an Auto Scaling Policy. Attach the policy to Auto-Scaling Group (ASG) and link ASG to ALB.
- Test by submitting a URL request and observe logs, ALB, and CloudWatch.
- Decide on the Scale-Out and Scale-In policies
- Mitigate failure





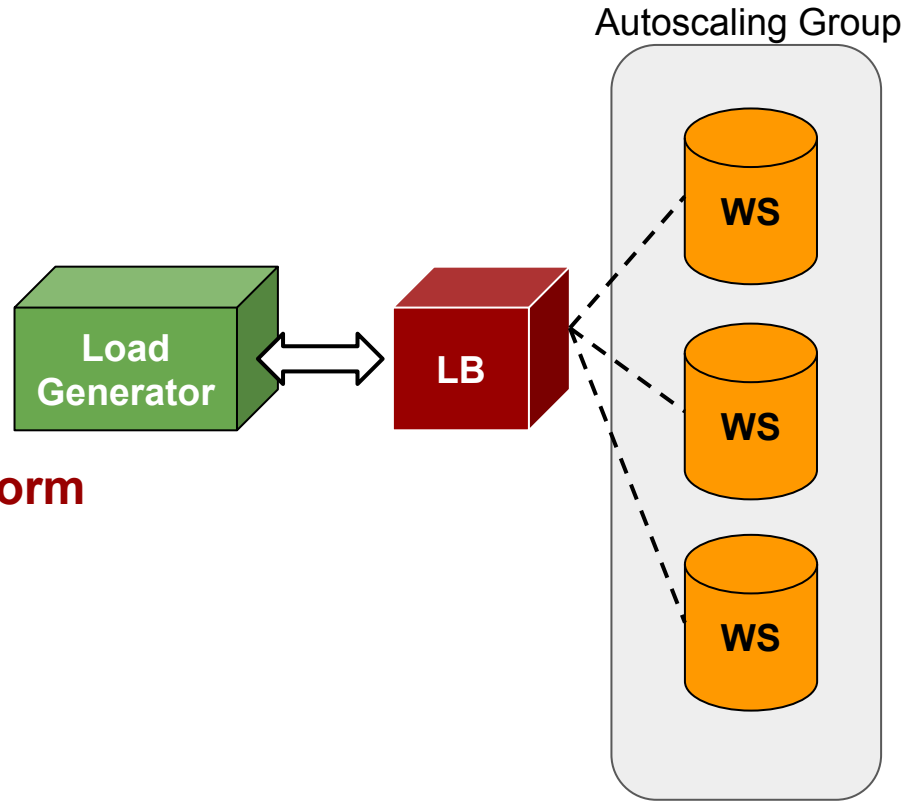
Hints for Project 2.1 AWS Autoscaling

Task 2 - AWS Auto Scaling

- Do a dry run via the console to make sure you understand the workflow completely and mimic that workflow programmatically.
- Autoscaling Test could be very expensive!
  - on-demand but now charged by the second
- Determine if there is a less expensive means to test your solution
- Creating and deleting security groups can be tricky
- CloudWatch and monitoring in ELB is helpful
- Explore ways to check if your instance is ready
- **Understanding the API documents could take time**

Project 2.1 Scaling on AWS

- Task 1
  - AWS Horizontal Scaling
- Task 2
  - AWS Auto Scaling
- **Task 3**
  - **AWS Auto Scaling with Terraform**



Project 2.1 Scaling on AWS

Task 3 - AWS Auto Scaling with Terraform:

- Read the Infrastructure as Code primer to learn about infrastructure automation
- Update your code to create and attach an IAM role to your Load Generator.
- Remove the code to launch the AWS resource required for autoscaling (ALB, Alarms, ASG…) resources (as they are in the TF config)
- Makes sure that **`terraform plan`** generates the expected resource

Project 2.1 Code Submission

- Submit the horizontal scaling task on AWS's load generator (LG) instance
- Submit the autoscaling task to the AWS load generator (LG) instance
- Submit the Terraform autoscaling task to the AWS load generator (LG) instance
    - You can use the test id from the AWS autoscaling task when creating the **.tar.gz** file
    - `mkdir testId; tar -zcf <testId>.tar.gz <testId>`
- **Remove all hidden files from your submission(.git / .idea / .DS_store)**
    - `export COPYFILE_DISABLE=true;` **(Mac Users)**

```
tar --exclude='.*' -zcf <testId>.tar.gz <testId>
```

Penalties for Project 2.1

Violation	Penalty of the project grade
Spending more than $20 for this project phase on AWS	-10%
Spending more than $35 for this project phase on AWS	-100%
Failing to tag all your resources in either parts (EC2 instances, ELB, ASG) for this project with the tag: key=Project, value=2.1	-10%
Submitting your AWS/Andrew credentials in your code for grading	-100%
Using instances other than t2.micro,t3.micro (testing only) or m5.large for Horizontal scaling on AWS	-100%
Using instances other than t2.micro ,t3.micro (testing only), m5.large for Autoscaling on AWS	-100%
Submitting executables (.jar, .pyc, etc.) instead of human-readable code (.py,.java, .sh, etc.)	-100%

Penalties for Project 2.1 cont.

Violation	Penalty of the project grade
Attempting to hack/tamper the autograder in any way	-200%
Cheating, plagiarism or unauthorized assistance (please refer to the university policy on academic integrity and our syllabus)	-200%

AWS Cloud APIs

- AWS CLI (link)
- AWS Java SDK (link)
- AWS Python SDK (link)

This Week

- **Quiz 4 (OLI Modules 5 & 6)**
  - Due on **Friday**, Feb 8th, 2019, 11:59PM ET
- **Project 2.1**
  - Due on **Sunday**, Feb 10th, 2019, 11:59PM ET
- **Primers released this week**
  - P2.2 - Intro to Containers and Docker
  - P2.2 - Kubernetes and Container Orchestration

Team Project - Time to Team Up

15-619 Students:

- Start to form your teams
  - See @5 for other potential team members
  - See @387 for suggestions on creating your post for @5
  - Choose carefully as you cannot change teams
  - Look for a mix of skills in the team
    - Web tier: web framework performance
    - Storage tier: deploy and optimize MySQL and HBase
    - Extract, Transform and Load (ETL)
- Create an AWS account only for the team project
- Wait for our post on Piazza to submit your team information

Questions?