

Developer Operations

Monitoring Cloud Services

You Are Flying Blind Without Instrumentation

- To use the cloud effectively, you need to know what is happening, e.g.
 - Are services up and running?
 - Are applications performing as expected?
 - Are there any faults or bottlenecks?
 - How much of your infrastructure is actually being used?



Collecting Metrics: Amazon CloudWatch



Amazon CloudWatch:

- Monitors instances, and collects and processes raw data into readable, near real-time metrics.
- Sends notifications and triggers auto scaling actions based on metrics you specify.

CloudWatch Monitoring

CloudWatch offers:

- A distributed statistics gathering system; it collects and tracks metrics.
- Metrics that are seamlessly collected at the hypervisor level by default.
- The ability to create and use custom metrics of data generated by your own applications and services.
- Examples:
 - The time web pages take to load
 - Request error rates
 - Number of simultaneous processes or threads

CloudWatch Alarm Examples

EC2



If CPU utilization is > 60% for 5 minutes...

RDS
Database



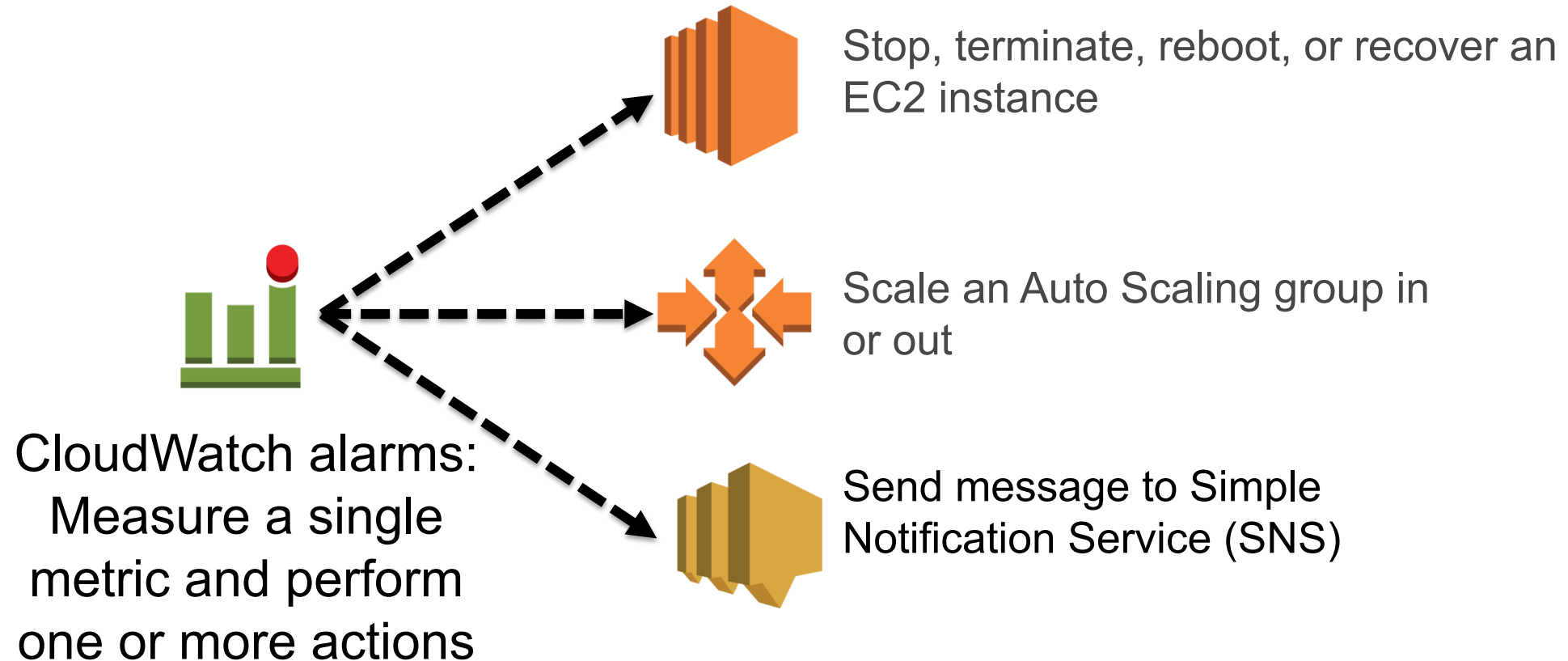
If number of simultaneous connections is > 10 for one minute...

Load
Balancer



If number of healthy hosts is < 5 for 10 minutes...

CloudWatch Alarms and Actions



Basic Custom Monitoring for Assignment 1

- We will use CloudWatch later in the semester (Assignment 2)
- For now, we will write our own simple monitoring tools with Python and use some basic DevOps to deploy these and ideally respond to what is observed
- Assignment 1 solution
 - Uses **local** Python program to launch EC2 instance and install web server
 - Copies **monitoring** program (also in Python) up onto instance and runs this **remotely** on the instance
 - This program is provided: `check_webserver.py`

check_webserver.py

```
import subprocess
```

```
def checkhttpd():
```

```
    try:
```

```
        cmd = 'ps -A | grep httpd'
```

```
        subprocess.run(cmd, check=True, shell=True)
```

```
        print("Web Server IS running")
```

```
    except subprocess.CalledProcessError:
```

```
        print("Web Server IS NOT running")
```

```
def main():
```


```
    checkhttpd()
```

```
# This is the standard boilerplate that calls the main() function.
```


```
if __name__ == '__main__':
```

```
    main()
```

Check for httpd process
by filtering output of **ps**
command



No exception thrown, so
grep must have found a
match in ps output



Exception thrown



Copy check_webserver.py up to EC2 instance

- Use scp command
- Test using terminal

```
$ scp -i keyname.pem check_webserver.py ec2-user@11.22.33.44:.
```

- Then do it from a Python script – e.g. using subprocess.run()

Run check_webserver.py remotely on EC2 instance

- Use ssh remote command execution
- First need to install Python3 on instance

```
$ ssh -i keyname.pem ec2-user@11.22.33.44 'sudo yum install python37'
```

- Then we can run the script

```
$ ssh -i keyname.pem ec2-user@11.22.33.44 'python3 check_webserver.py'
```

- Alternatively we can run directly, but need to give it permissions

```
$ ssh -i keyname.pem ec2-user@11.22.33.44 'chmod 700 check_webserver.py'
```

```
$ ssh -i keyname.pem ec2-user@11.22.33.44 './check_webserver.py'
```

Exercises

- See additional week 5 monitoring exercises