Monitoring

Overview

- What does software monitoring consist of?
- Why do we monitor software?
- Monitoring infrastructure
- Monitoring applications

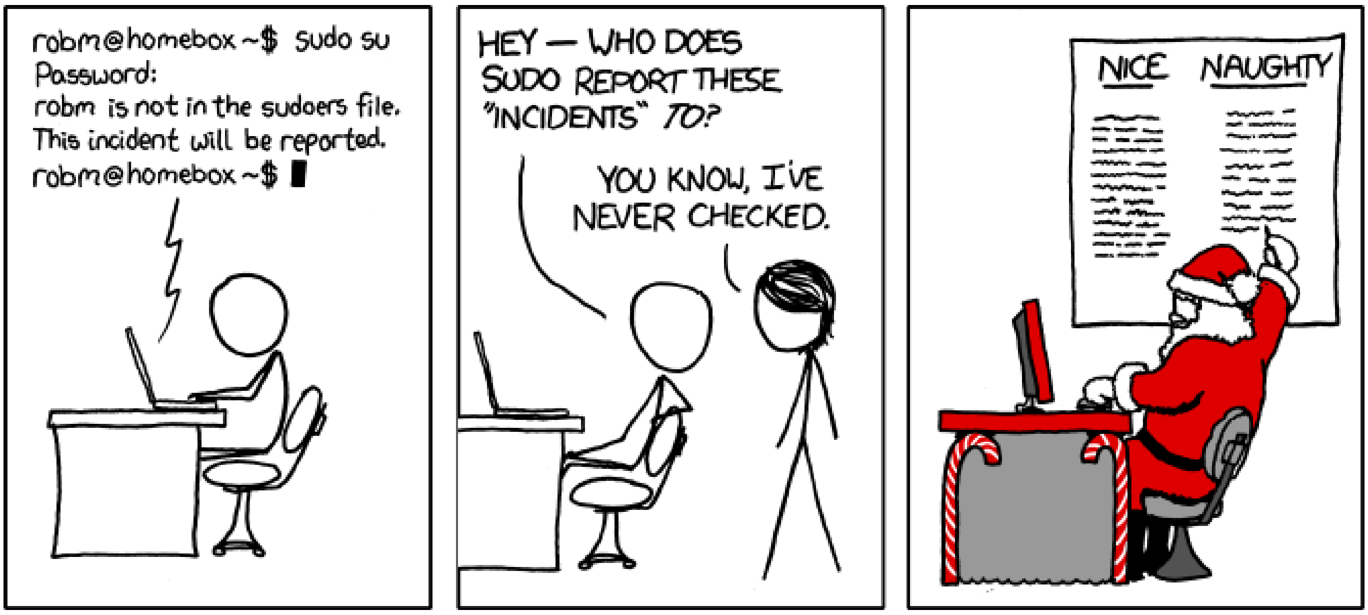
Learning Objectives

- Identify the features software monitoring
- Explain why and what we monitor
- Define logging and what logs are used for
- Detail the features of Cloudwatch and Grafana

Monitoring our software

Monitoring software involves continuously examining a running system with two main aspects

1. Monitoring and measuring the performance of the system
2. Monitoring and auditing events that occur within the system


robm@homebox~$ sudo su
Password:
robm is not in the sudoers file.
This incident will be reported.
robm@homebox~$
HEY — WHO DOES SUDO REPORT THESE "INCIDENTS" TO?
YOU KNOW, I'VE NEVER CHECKED.
NICE
NAUGHTY

What do we track?

What kind of things do we track?

- Uptime
- Resource utilisation (CPU, memory, network)
- Performance (requests/s, ops/s)

- Errors
- Business KPIs
- User interactions

These are several of things that we can monitor in a running system. Can you give any other examples of things we might track?

Why do we monitor?

- To take action as soon as (or even before) things go wrong
- Understand actual application usage patterns
- Make more accurate business decisions
- Learn what normal looks like
- Assess the impact of our releases: performance degradation, regressions, downtime...
- Identify areas in our architecture to improve
- Make on-call work humanly possible

Be proactive rather than reactive when it comes to managing the health of your application. As full-stack, DevOps engineers deploying stuff to prod is only the beginning. The service needs to be supported and managed afterwards. Identify pinch points. Also, on the release impact point, it allows us not only to smoke test our app post-release but also pre-release as we can deploy the same monitoring config to our test environments Regression: a software bug that makes a feature stop functioning as intended after a certain event (e.g. system upgrade)

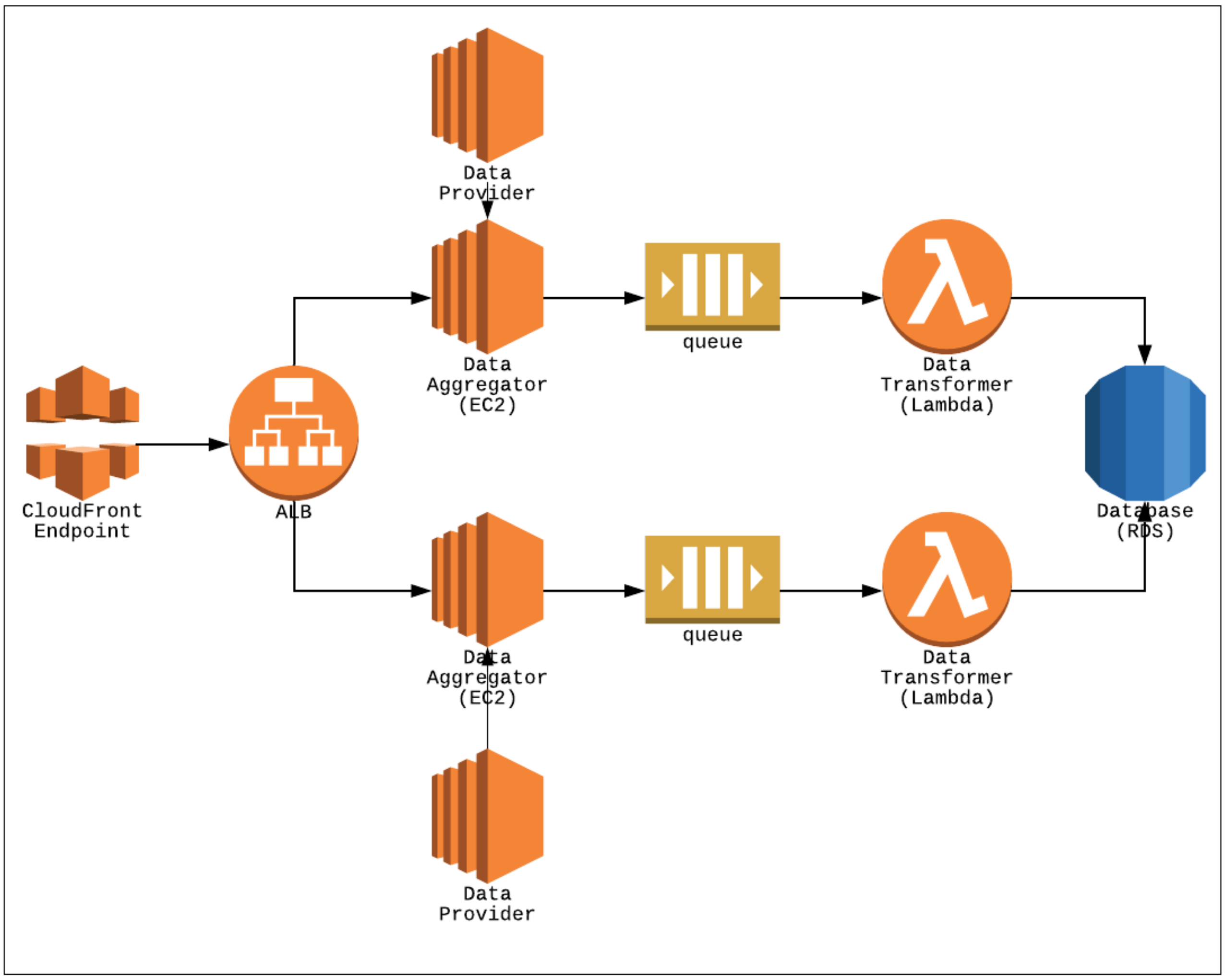
What do we monitor?

- Infrastructure
- Applications
- Component interactions

Component interaction monitoring is very important with distributed architectures, like micro-services. We won't talk about this one today, but this would be tools like X-Ray, Jaeger or service mesh built-ins, like istio & kali

---

What can we monitor?



Point out components of this diagram that are worthwhile monitoring in some way: individual elements (hardware, infra), apps running in the EC2s and Lambdas, as well as communication channels (X-Ray, etc.)

Monitoring Infrastructure

CloudWatch

Cloudwatch is a monitoring and alerting managed service for AWS and offers the following:

- Log aggregation, retention, rotation and search
- Monitoring dashboards for AWS infrastructure out-of-the-box
- Alarms and system events

We will expand on CloudWatch shortly.

SNS

Pushes out notifications to different subscribers via various channels, e.g.

- Email

- SMS
- Mobile push notifications
- Lambda

SNS - Concepts

Topic: notification category. You or an application can subscribe to it and get notified when a new event gets pushed to it

Subscription: sets the receiver of a notification and also dictates how they'll receive it

---

Monitoring applications

Application Logs

A **log** is a message produced by a running application capturing some noteworthy information about its state or an event taking place at that point in time

You can think of logs as your app's running journal

Logging Levels

Logging statements are added at key areas of your codebase and can serve many different purposes

i Info: they document normal state changes in the app. They contain non-actionable information

⚠ Warning: non-fatal errors, the kind of events you can leave until the next morning

🚨 Error: fatal, the kind of events you'd be woken up in the middle of the night to resolve

Log Levels help to show how important an event is when you write your program. When you write your programs, you may also use the DEBUG log level.

What do we use logs for?

- Obtaining failure messages when our app crashes
- Tracing application flow
- Debugging our running application
- App health monitoring
- Timing stuff precisely

Failure messages from your system can be generated by your log messages. These can also be generated by the underlying system, if it encounters an error that you have not coded for.

How do you log?

At it's simplest - write the message to stdout!

```
# Python
try:
    div = 1 / 0
except Exception as e:
    print("ERROR: " + str(e))
```

No, really, how do you log

Most languages now have logging libraries available.

These remove much of the boilerplate and setup required to produce manageable and standardised logs.

```
# Import the built-in logging lib
import logging

# Create an app logger
logger = logging.getLogger(__name__)

try:
  div = 1 / 0
except Exception as e:
  # Log an exception, it will go to stdout by default
  logger.exception(e)
```

Where do logs go?

By default all logs will be printed to stdout and stderr (usually terminal output) and will therefore disappear after a short while.

If we want to store our logs for future reference we need to send them to some persistent storage. eg. a file

```
# Redirect stdout to a file
$ python app.py > app.log

# Redirect stdout and stderr to a file
$ python app.py &> app.log
```

stdout is useful if you are working directly on a program on what to know what is happening.

stdout is not useful for long running systems where hundreds or thousands of log messages can be generated every hour, especially if there was an issue last night and you're looking at it a day later.

Storing Logs

Most applications have a default location where they store their log files, although many will allow you to set the location.

On Unix systems that location is usually `/var/log`.

The app will keep appending logs to its log file as long as it's running.

This has a couple of downsides...

Downsides of log files

- Log files will keep growing if you keep logging
- Log files sent to stdout are only available on the server that produced them
- Log files are hard to search and query

These are problems that occur when setting logs, although all of them have been solved. Can you come up with any ways in which that might have happened.

Growing logs - rotating logs by hour/day Local log files - log shipping to other servers Hard to search - formatted log files and log parsers.

AWS CloudWatch

What is CloudWatch?

- Allows you to monitor AWS applications in near-real-time
- Automatically configured to provide metrics on request counts, latency, and CPU usage
- Can also send your own logs and custom metrics to CloudWatch for monitoring
- Keep track of your application performance, resource use, operational issues, and constraints
- Helps organizations resolve technical issues and streamline operations

---

CloudWatch Logs

CloudWatch can collate and store all of your logs for as long as you want so you don't have to worry about maintaining them.

Some AWS managed services, like Lambda, will forward all logs automatically to CloudWatch, where you'll be able to search and query them.

CW Logs Example

CloudWatch Logs Simple Queries

CloudWatch logs can be queried by filtering on specific parameters:

```
{ $.level="info" && $.event="app.init" }
```

- `level` refers to the log level is set by your logger of choice. Examples include `error`, `warning`, `info` and `debug`.
- `event` refers to the function that raised the log.

You can also query by dates and other factors.

Query example

CloudWatch Logs Insights

CloudWatch Logs Insights is a more advanced and central part of the AWS monitoring ecosystem. You can use Log Insights to search and analyze your log data interactively. It enables you to query your logs and can assist you in responding to operational issues.

The below query demonstrates selecting a set number of fields, applying a filter and then applying a sort. This is useful if you need to sift through 100s or 1000s of logs.

```
fields event, level, message
| filter level = "info" and event = "app.init"
| sort @timestamp
```

CloudWatch Metrics

Metrics are the fundamental concept in CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch.

Think of a metric as a variable to monitor, and the data points as representing the values of that variable over time.

For example, the CPU usage of a particular EC2 instance is one metric provided by Amazon EC2. The data points themselves can come from any application or business activity from which you collect data.

Monitoring AWS Lambda

CloudWatch monitors your Lambda applications automatically. All you need to do is create one and CloudWatch will start tracking metric data for it.

Some of the core metrics that it gathers on Lambda are:

- **Invocations**: The number of times your function is invoked.
- **Errors**: The number of times your function fails with an error, due to timeouts, memory issues, unhandled exceptions, or other issues.
- **Throttles**: The number of times your function is throttled. AWS limits the concurrent number of executions across all your functions. If you exceed that, your function will be throttled and won't be allowed to run.
- **Duration**: How long your function runs.

CloudWatch Alarms

An alarm watches a single metric over a specified time period, and performs one or more specified actions, based on the value of the metric relative to a threshold over time.

You can use an alarm to automatically initiate actions on your behalf.

The action is a notification that can sent to a service like Simple Notification Service (SNS) or a monitoring dashboard.

Alarms invoke actions for sustained state changes only. Alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods. An example is the CPU usage of an EC2 instance is over 90% for 5 minutes.

Speaking of monitoring dashboards...

Grafana

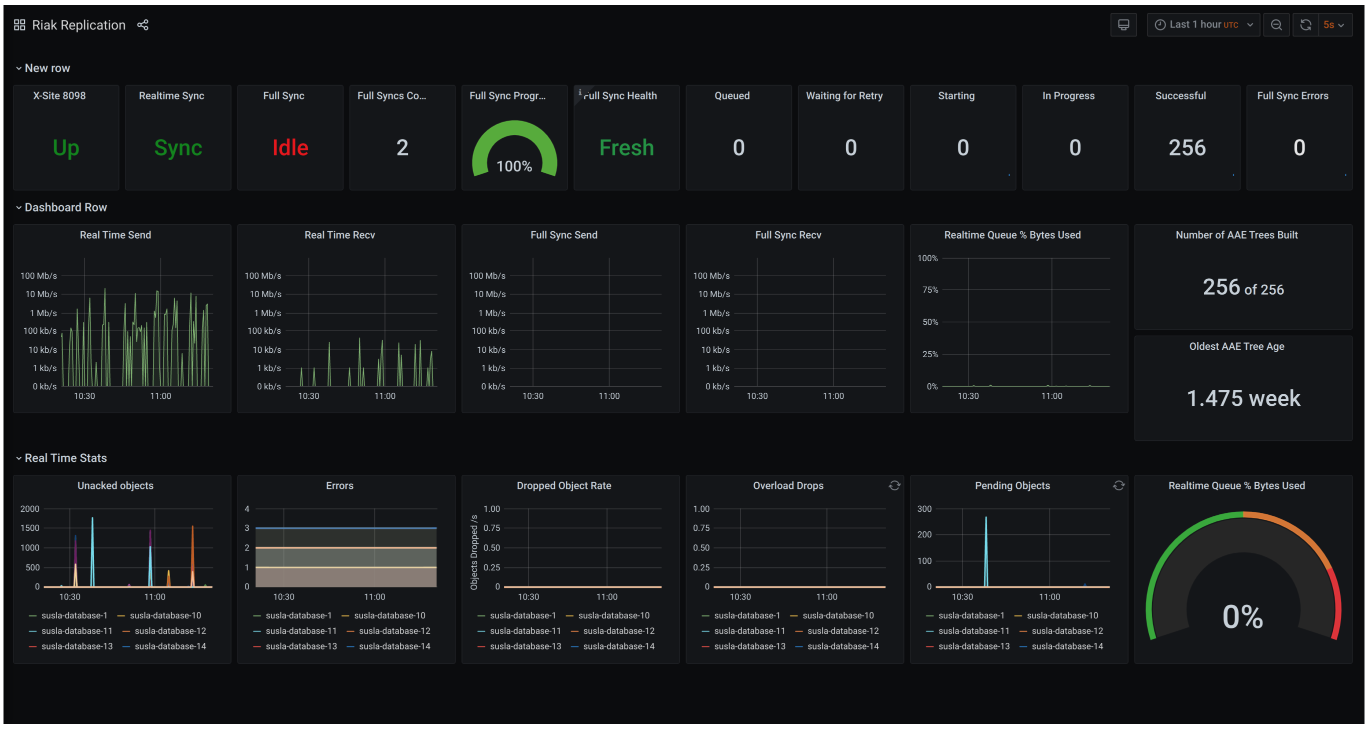
What is Grafana?

Grafana is an open source monitoring dashboard application with lots of features which make it easy to use, and is both flexible very powerful!

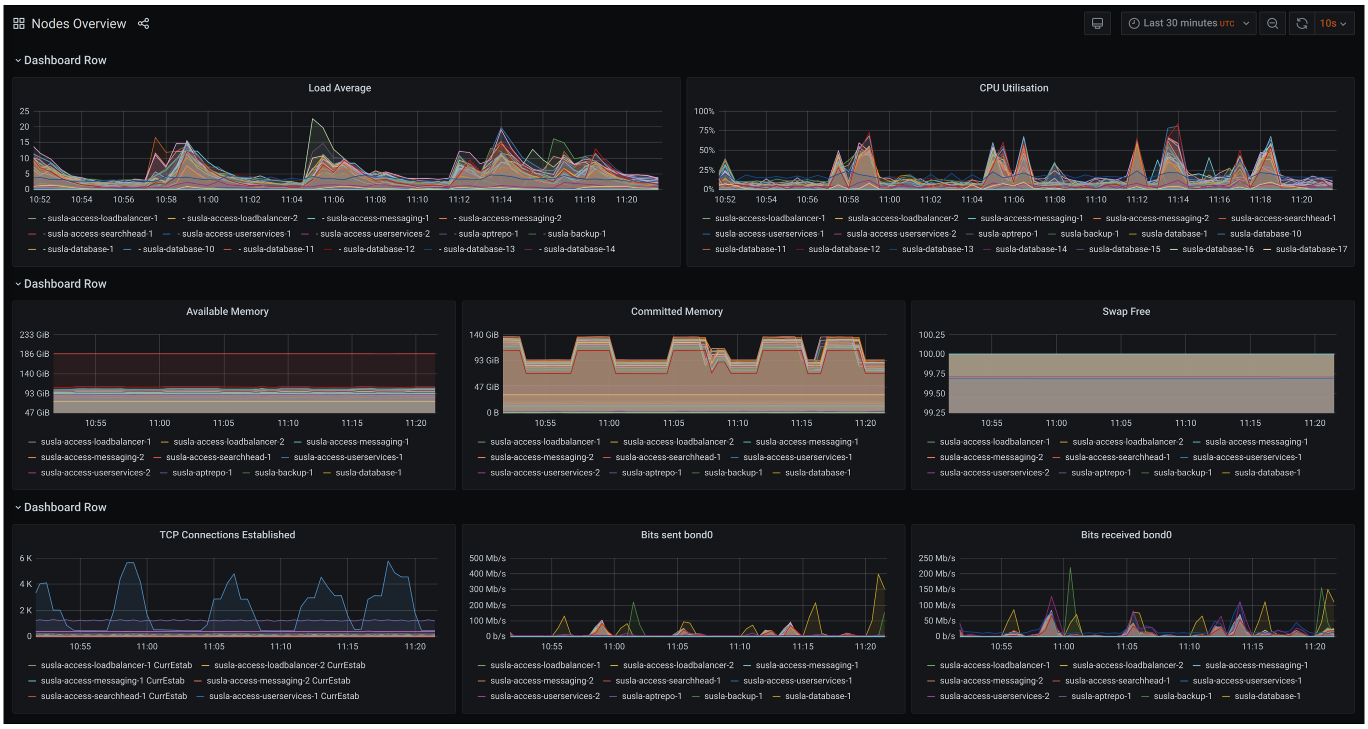
It pulls in data from supported data sources to create dashboards including databases and CloudWatch metrics.

Grafana can pull in data from a wide range of sources and create dashboards from them.

What does it look like?



What does it look like?



Running Grafana

Grafana is very easier to start and work with.

```
$ docker run -d -p 3000:3000 grafana/grafana
```

Then open a browser window on http://localhost:3000 to see the Grafana front page.

Once the learners have opened the window, run the random data generation and display it in a dashboard. The user/pass is admin/admin. There is a later exercise to set up and use Grafana with the local TestDB.

---

docker-compose

Or you can use a docker-compose.yml file for added configuration:

```
version: "2"
services:
  grafana:
    image: grafana/grafana:5.4.3
    ports:
      - "3000:3000"
    volumes:
      - grafana:/var/lib/grafana
volumes:
  grafana:
```

Exercise

Instructor to distribute exercises.

Learning Objectives Revisited

- Identify the features software monitoring
- Explain why and what we monitor
- Define logging and what logs are used for
- Detail the features of Cloudwatch and Grafana

Further Reading and Credits

- CloudWatch Docs
- CloudWatch Logs Insights Query Syntax
- CloudWatch Logs Query Syntax
- Grafana Docs