Logging, monitoring, alerts and status pages

Logging refers to the practice of recording events, messages, or data points generated by software applications and systems. These logs can include various types of information, such as:

- **Error Messages:** Details about errors and exceptions that occur.
- **Access Logs:** Records of who accessed what resources and when.
- **Audit Logs:** Records for compliance and security purposes.

Monitoring

Monitoring involves the continuous observation of a system to ensure it is operating correctly and efficiently. It includes tracking various metrics and performance indicators, such as:

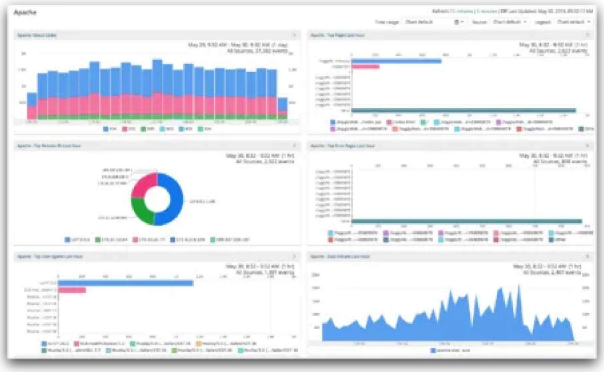
- **CPU Usage:** Monitoring how much processing power is being used.
- **Memory Usage:** Tracking the amount of memory being utilized.
- **Disk I/O:** Observing read/write operations on storage devices.
- **Space:** Total amount of space available on the machine
- **Network Traffic:** Measuring data transfer rates and network activity.
- **Application Performance:** Monitoring response times, throughput, and error rates.

Alerts

When there are logging/monitoring systems in place, you can put up alerts to be called/messaged/slacked/paged when

1. A system goes down
2. CPU usage goes above a certain point
3. Error count goes up

Monitoring dashboards



Status pages

https://status.100xdevs.com/

Services

There are a few ways to monitor systems.

Paid

1. **AWS CloudWatch:** Monitoring and observability service for AWS resources.
2. Datadog - Logging and monitoring
3. Newrelic - Logging and monitoring

In house/self hosted

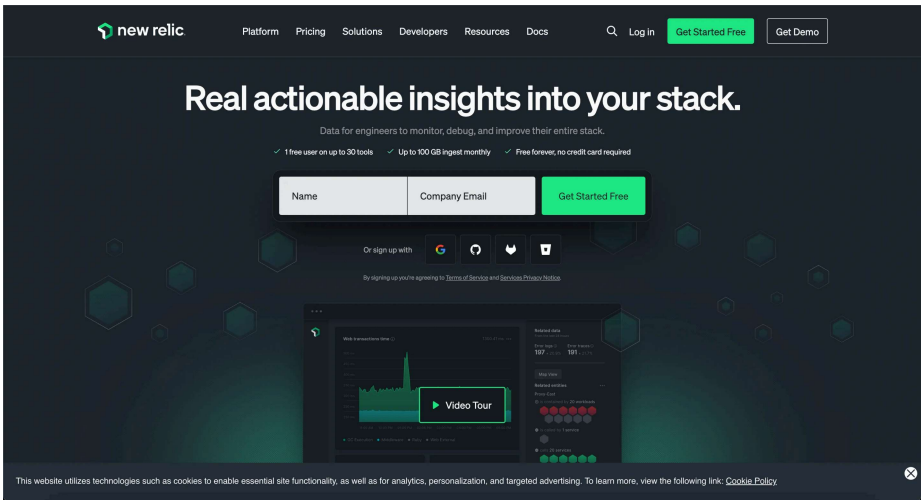
Prometheus + Grafana + Loki

We'll be going through

1. Newrelic
2. Prom + Grafana

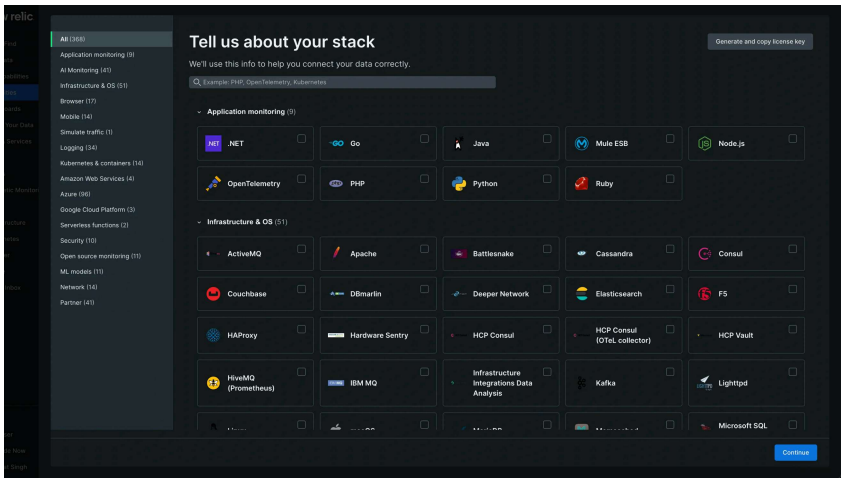
We'll be setting up 3 things in newrelic

1. Logging
2. Monitoring
3. Alerting

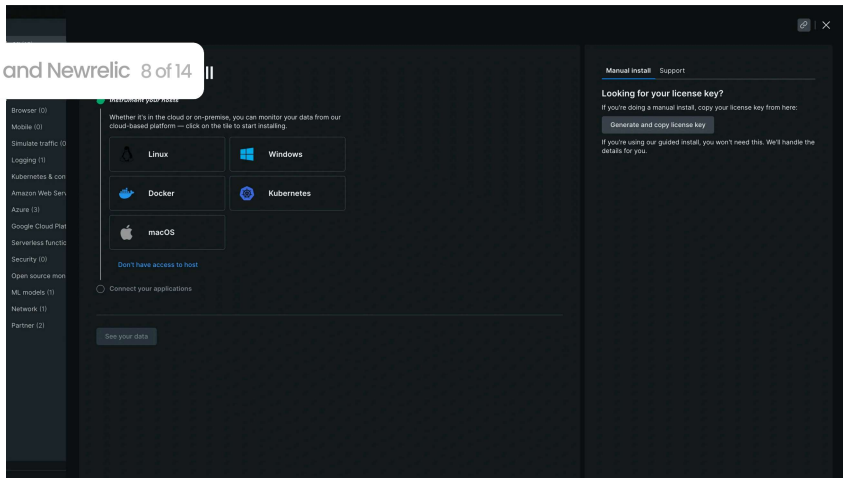


Steps

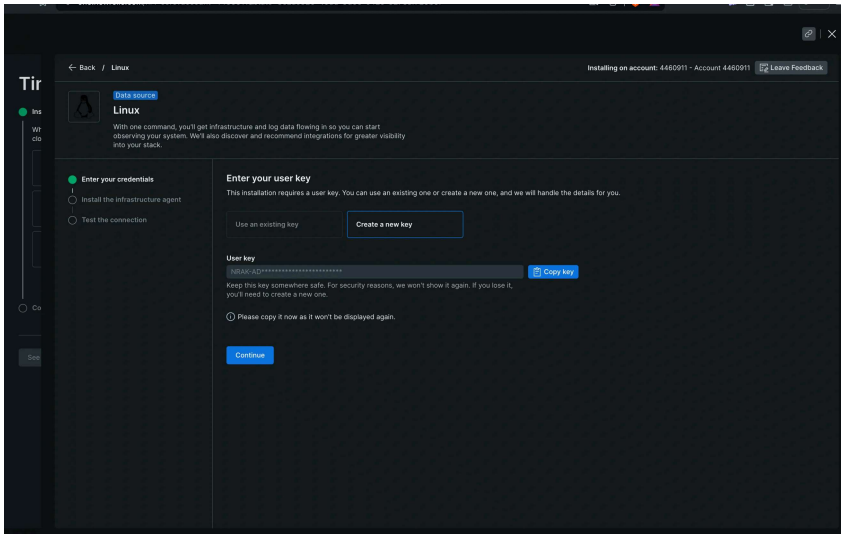
- Login to newrelic
- Select your stack



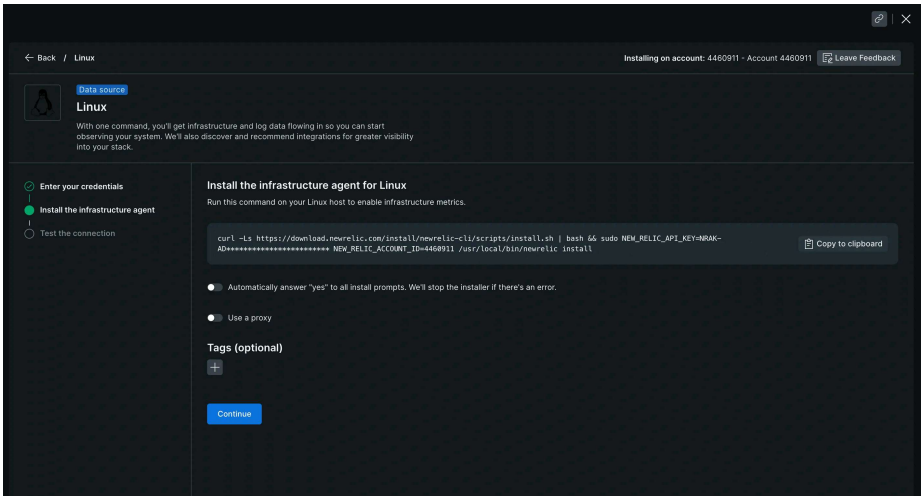
- Select machine (OS) to install it on



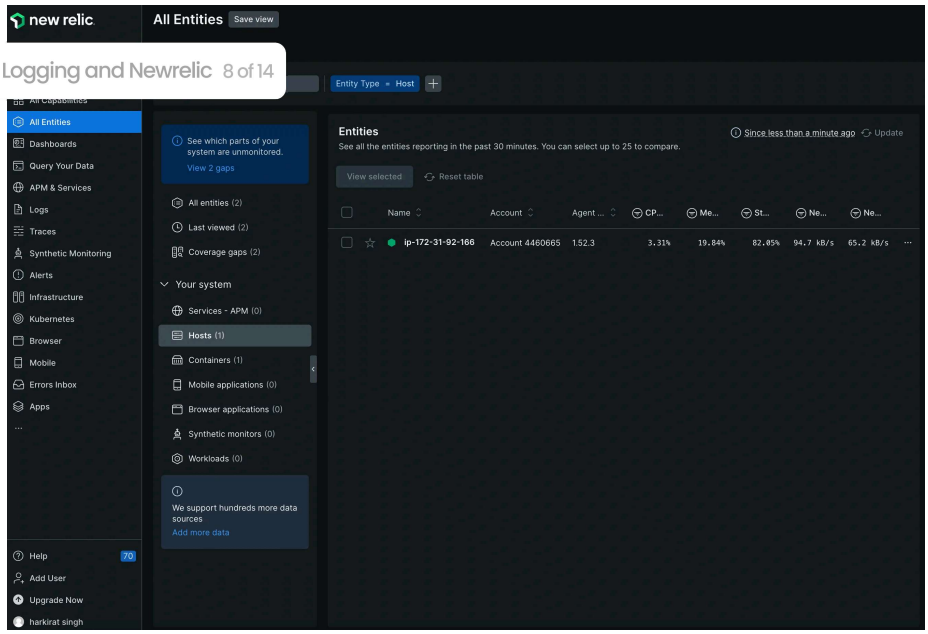
- Create a new key



- Run the command on your machine



Entities in newrelic



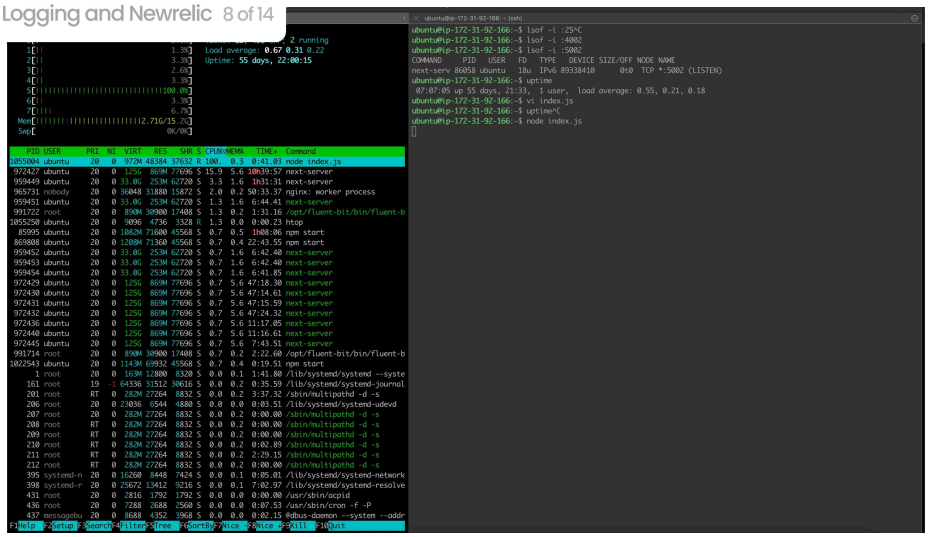
Gives you all oberservability on the host

Experiment

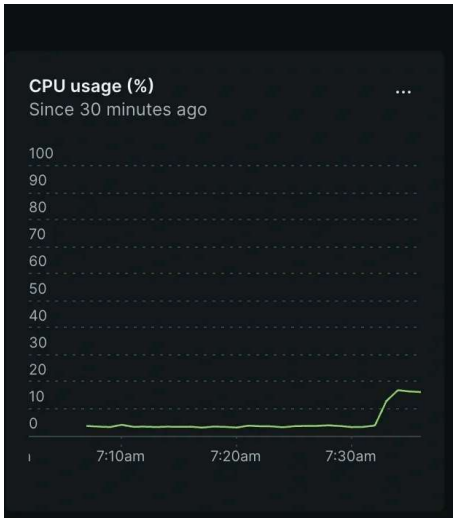
Add an index.js file with an infinite loop

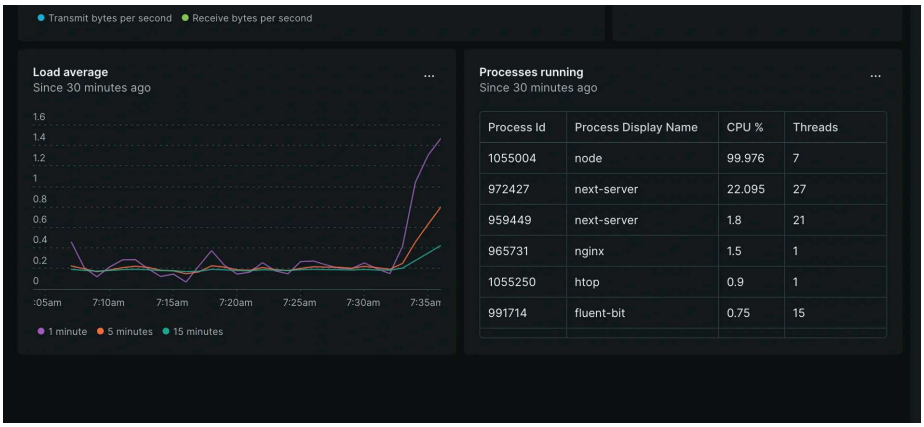
```
let c = 1;
while(1) {
  if (Math.random() < 0.1)
  c = c + 1;
}
```

On the machine



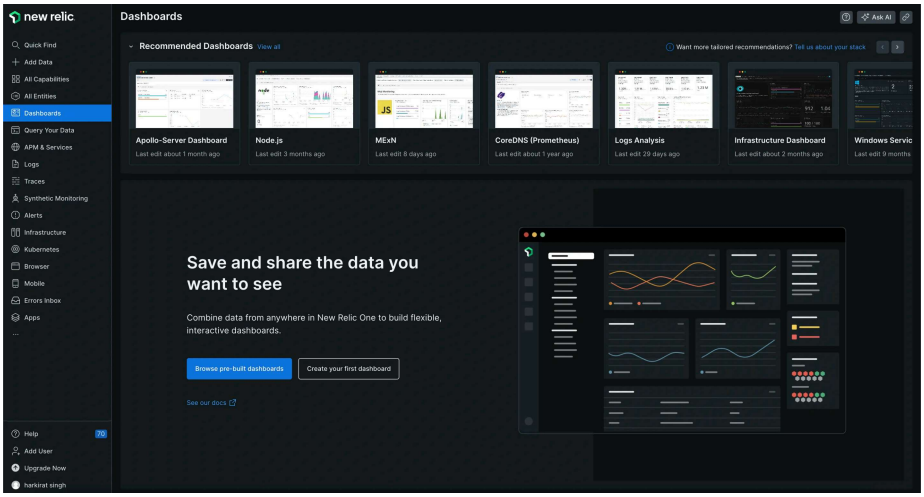
On newrelic



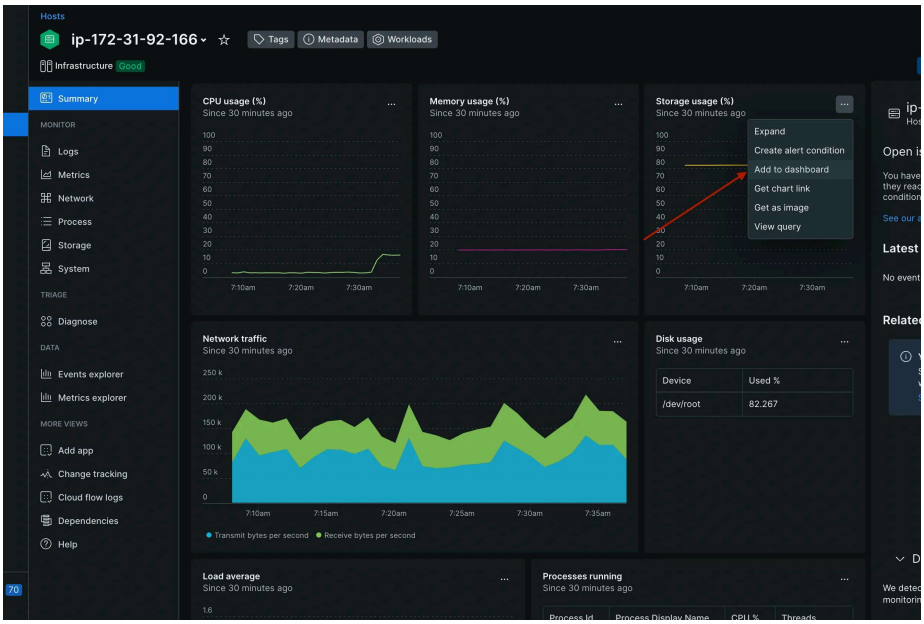


Dashboards

https://one.newrelic.com/dashboards

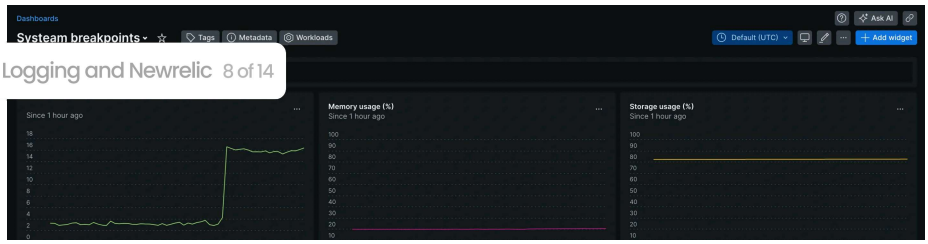


Adding widgets to a dashbaoard



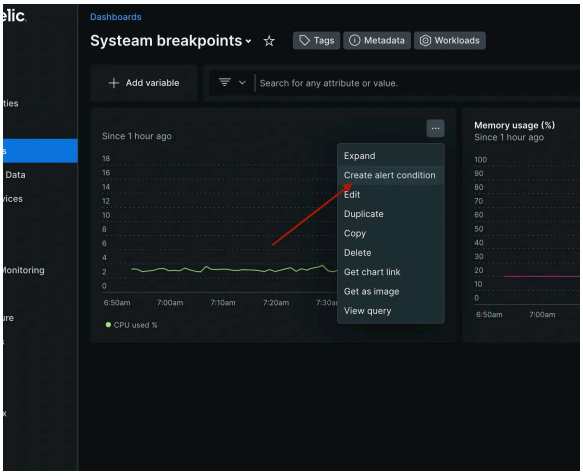
Assignment

Try to create a dashboard similar to the following -

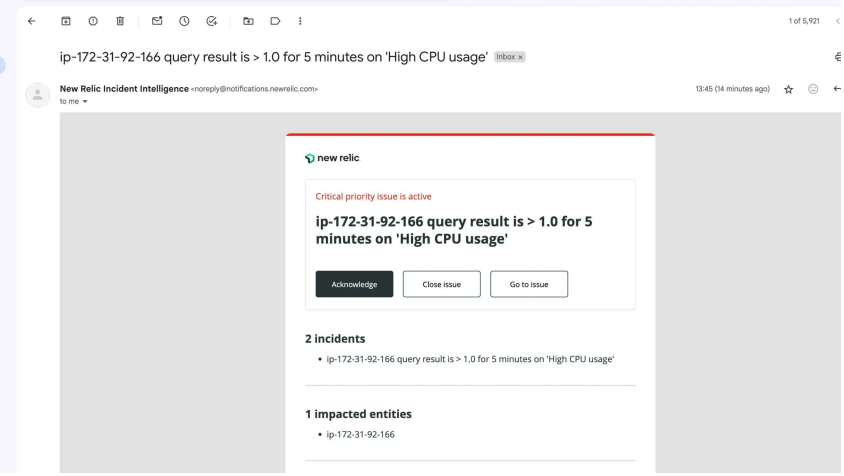


Alerts

The goal of monitoring is to make sure if there are spikes in some metrics, users get informed via call/messages/slack



1. Create an alert
2. Attach it to a policy
3. Add a notification to the policy (email, slack...)



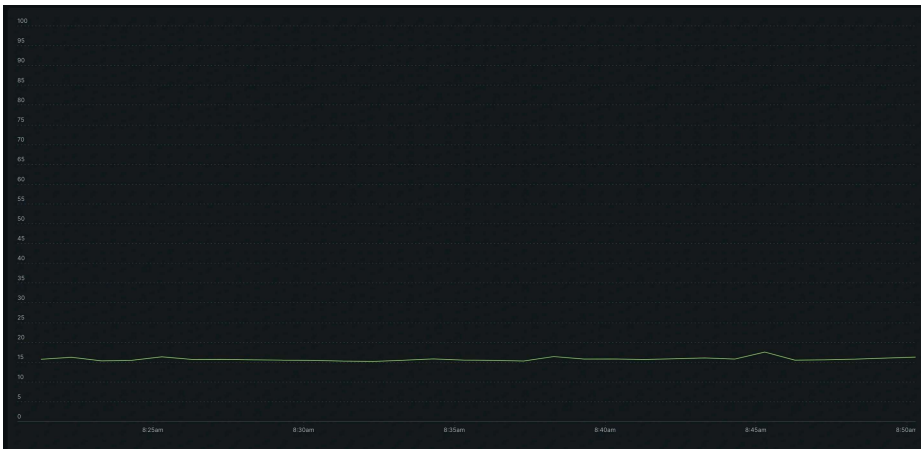
NRQL

NRQL (New Relic Query Language) is a SQL-like query language used to query data within New Relic, a monitoring and observability platform. NRQL allows users to perform complex data analysis and create custom dashboards and alerts by querying their application and infrastructure performance data stored in New Relic.

CPU Usage

- Fetch all CPU usage

```
SELECT average(cpuPercent) AS `CPU used %` FROM SystemSample WHERE (en
```

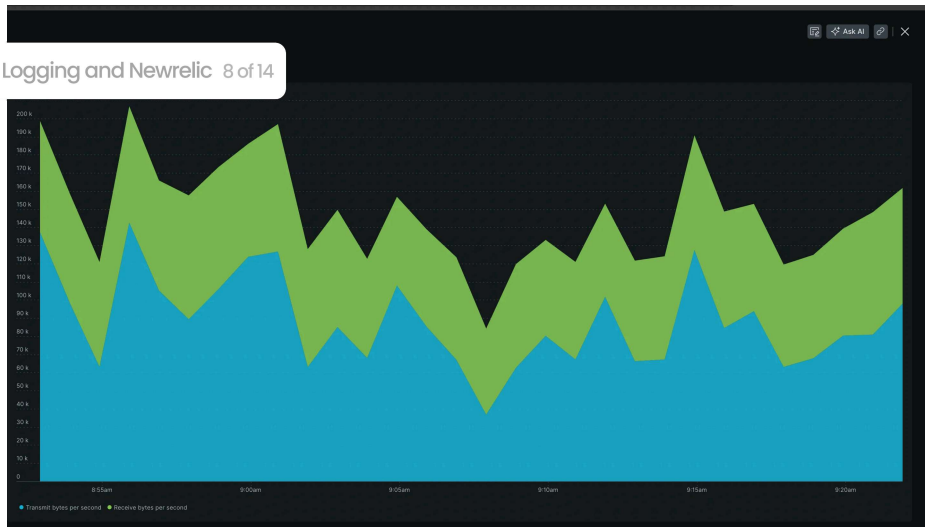


- Filter by High CPU Usage

```
SELECT average(cpuPercent) AS `CPU used %`
FROM SystemSample
WHERE (entityGuid = 'NDQ2MDY2NXxJTkZSQXxOQXwzOTczMjM4ODc0NzI4NDk0N
TIMESERIES AUTO
```

- Multiple graphs in the same timeline

```
SELECT average(transmitBytesPerSecond) AS `Transmit bytes per second`, ave
```



- Facets

```
SELECT average(cpuPercent) AS `CPU used %`
FROM SystemSample
WHERE entityGuid IN ('xyz', 'abc')
FACET entityGuid
TIMESERIES AUTO
```

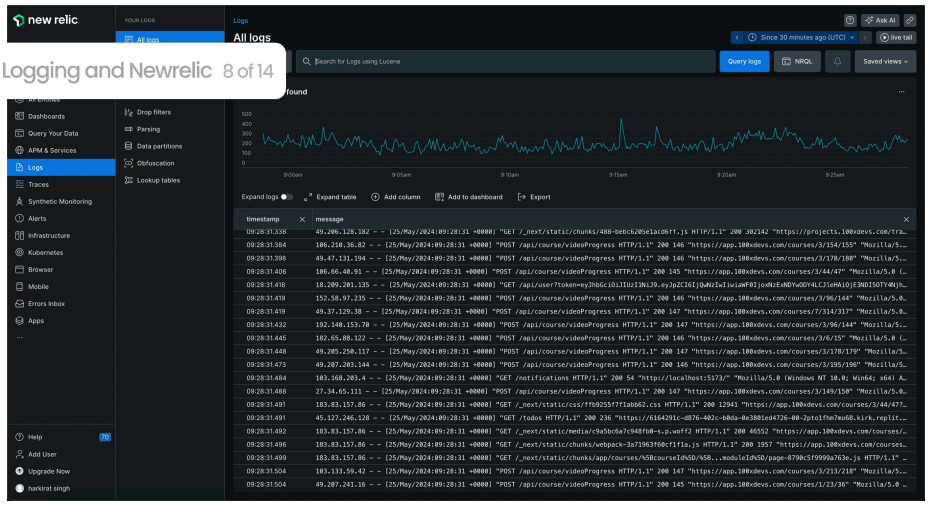
APM

Newrelic also lets you see all the logs and stats from your process, not just the host machine

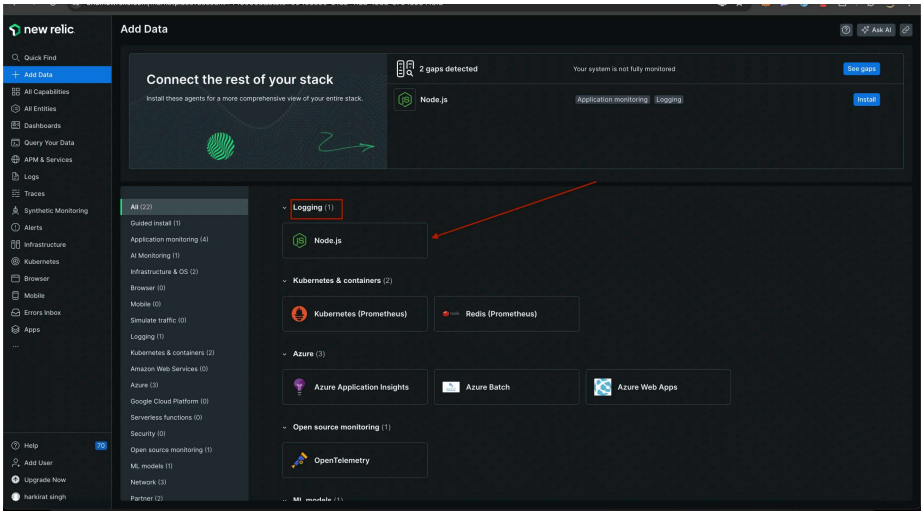
Default logs

By default, you will see it is tailing global nginx logs, but not the application logs

```
/var/log/nginx/access.log
```



Setting up APM for a Node.js process



- Create a new data source with Node.js
- Use one of 5 ways to set it up (we'll go with on the host)
- Install dependencies

```
npm install newrelic express
```

- Update package.json

```
"scripts": {
  "start": "NEW_RELIC_APP_NAME=test NEW_RELIC_LICENSE_KEY=license_key n
},
```

- Create a simple express app

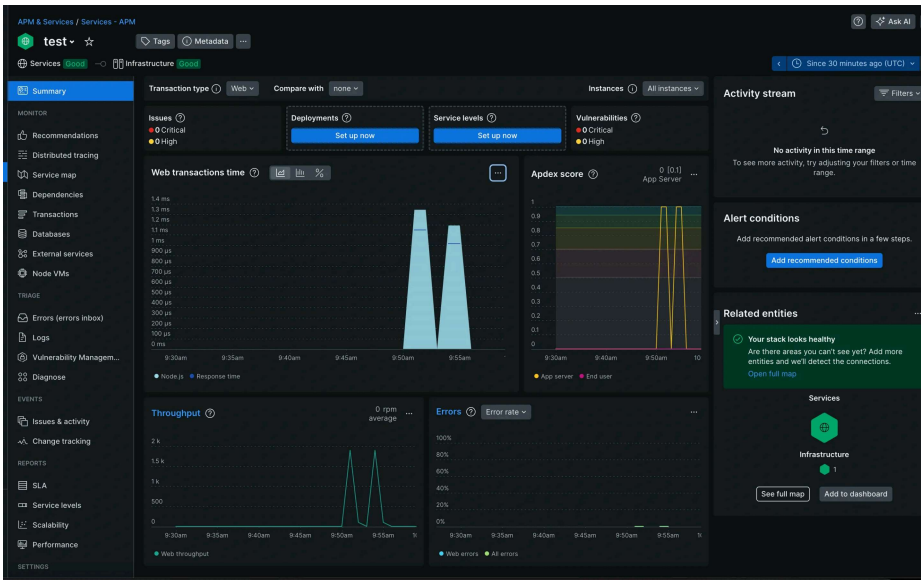
```
require("newrelic");

const express = require("express");

const app = express();

                    ;) => {
```

```
    console.log("route hit");
    res.send({ message: "hi there"})

app.listen(3000, () => {
  console.log("listening on port 3000");
});
```

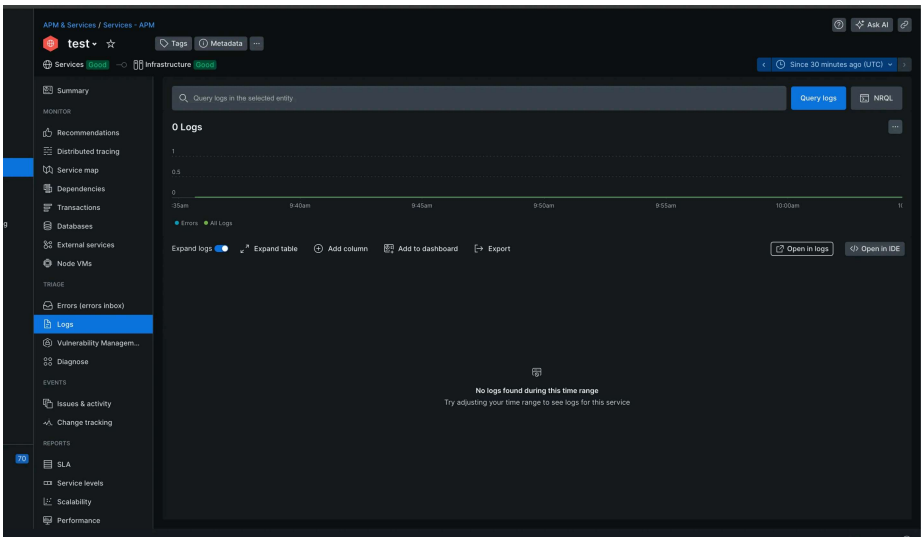
- Open another terminal and loadtest the app

```
npm i -g loadtest
loadtest -c 10 --rps 200 http://localhost:3000/
```

- Check the APM dashboard



- You will notice logs are still empty



Adding logs

Ref - https://docs.newrelic.com/docs/logs/logs-context/configure-logs-context-nodejs/

- Try enabling the NEW_RELIC_APPLICATION_LOGGING_FORWARDING_ENABLED

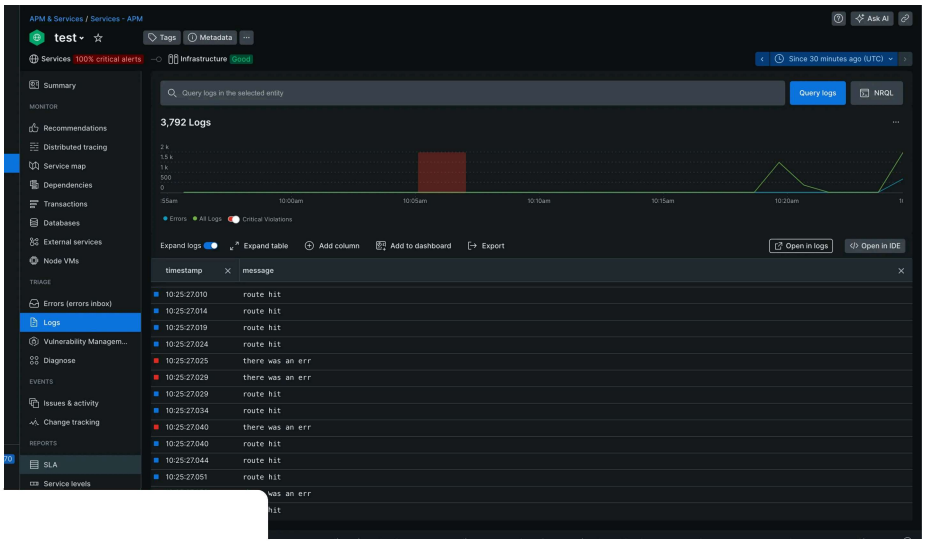
```
"scripts": {
  _APPLICATION_LOGGING_FORWARDING_ENABLED=true NE
},
```

- Add winston as the logger

```
npm install winston
```

- Update the code

```
require("newrelic");
const winston = require('winston');
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  defaultMeta: { service: 'user-service' },
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' }),
  ],
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.simple(),
  }));
}

const express = require("express");
const app = express();

app.get("/", (req, res) => {
   logger.info("route hit");
   if (Math.random() < 0.5) {
      logger.error("there was an err");
   }
   res.json({message: "hi there"})
});

app.listen(3000, () => {
   console.log("listening on port 3000");
});
```



Winston logger

Transports

Winston lets you stash logs in various places (files, console, postgres tables etc)

```
const logger = winston.createLogger({
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'combined.log' })
  ]
});
```

Good list of transports - https://github.com/winstonjs/winston/blob/master/docs/transports.md#postgresql-transport

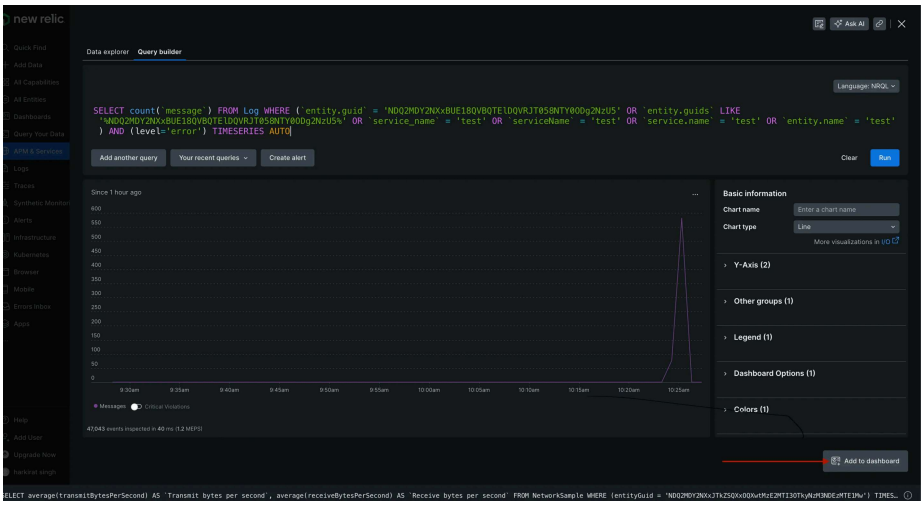
Formats

```
const { createLogger, format, transports } = require('winston');
const { combine, timestamp, label, prettyPrint } = format;

const logger = createLogger({
  format: combine(
    label({ label: 'right meow!' }),
    timestamp(),
    prettyPrint()
  ),
  transports: [new transports.Console()]
})

logger.log({
  level: 'info',
  message: 'What time is the testing at?'
});
// Outputs:
// { level: 'info',
//   message: 'What time is the testing at?',
//   label: 'right meow!',
//   timestamp: '2017-09-30T03:57:26.875Z' }
```

Metrics on logs

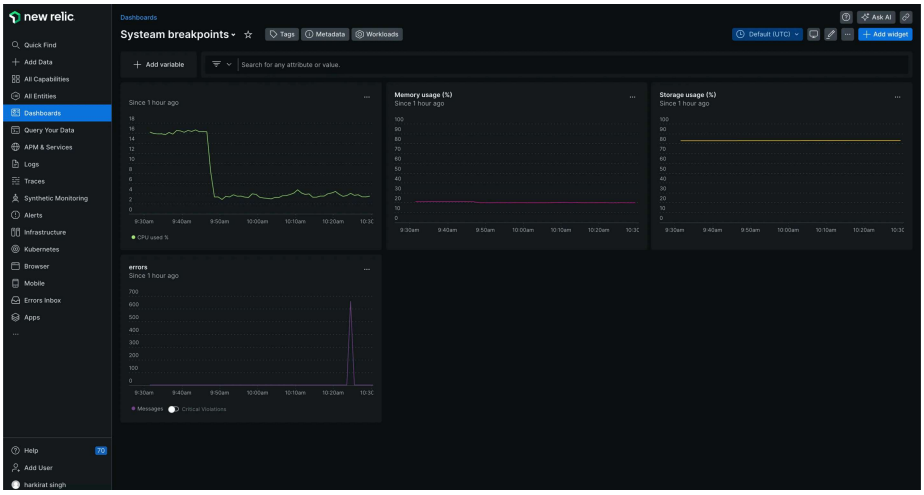
You can add metrics on top of log counts (esp for errors) to catch if a certain error is being thrown too often/there is a spike

```
SELECT count(`message`) FROM Log WHERE (`entity.guid` = 'NDQ2MDY2NXxBUE18
```



You can also add individual set of metrics on individual errors

```
AND message LIKE '%there was an error%'
```



p95, p99

When you are measuring things like response times , cpu usage etc, which of the following should you use?

1. Mean
2. Median
3. Something else?

Example

Let's say in a 20 second interval there were 20 requests that came and the response times were -

[1, 2, 3, 4, 4, 4, 5, 6, 7, 9, 10, 11, 11, 11, 12, 13, 13, 13, 50, 100]

Average - 14.45

Median - 9.5

Better metrics

As you can see, the average is skewed a little to the right because of two anomolies (50, 100)

This means the website is performing good for 90% of the users, but 10% of the users are having slower response times.

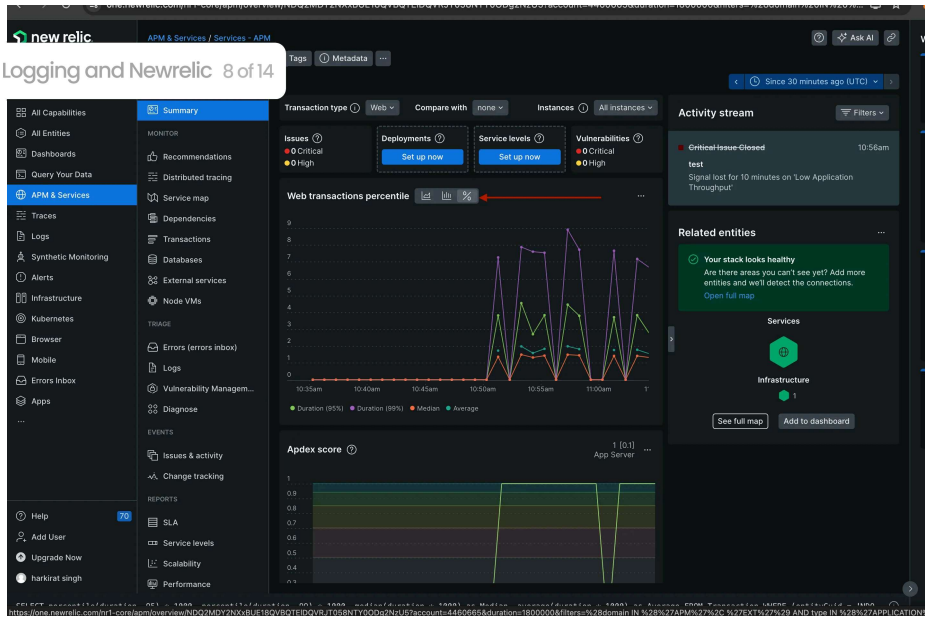
- P90 here is 13, since 90% of the users are equal to or below 13ms

▼ Can you guess what P95 would be?

P95 is 50, since 95% of the users are equal to or below 50ms

Percentile based metrics in newrelic

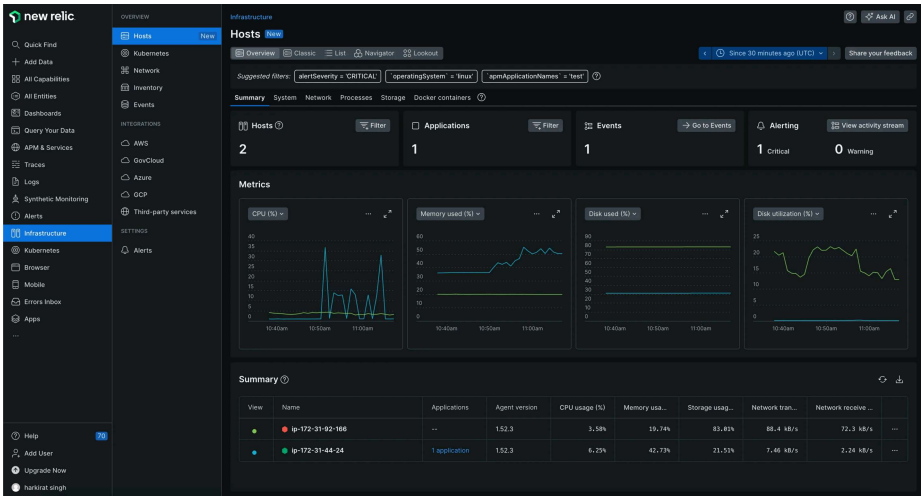
If you open the APM dashboard, you'll see the response time percentiles tab



Try exploring the query -

```
SELECT percentile(duration, 95) * 1000,
percentile(duration, 99) * 1000,
median(duration * 1000) as Median,
average(duration * 1000) as Average
```

Infrastructure tab



Status pages/Notifiers

Pagerduty and Better stack are two popular services used by teams for on call management, raising a call to the manager incase the on call is asleep

https://betterstack.com/uptime

Problem?

All services we've used (newrelic, uptime, datadog) can get very expensive very quickly.

You also become highly dependant on them as time goes by so very hard to get the costs down.

Solution

Use open source tools that let you do the same thing and self host.

Most common stack for monitoring, observability and logging - Grafanna, Prometheus and Loki

1. https://github.com/grafana/grafana
2. https://prometheus.io/
3. https://github.com/grafana/loki