mezmo.com

mezmo

MEZMO EBOOK

Debugging in Development with Mezmo

Mezmo gives users access to the information they need to effectively debug during development.

INTRODUCTION

Traditionally, logging was most commonly associated with the post-deployment part of the software development lifecycle, or SDLC. Logs typically served first and foremost to help IT engineers find and troubleshoot problems that arose in production.

Today, however, logging can help teams optimize much more than just production-environment application management. And indeed, logging needs to be leveraged across all stages of the SDLC in order to ensure the reliable, continuous delivery of software. Developers, testing teams, and anyone else involved in software delivery must make use of logs and log analysis as one way to ensure the smooth flow of code across the entire SDLC.

With that reality in mind, we've prepared this guide to showcase practical approaches to log analytics at different stages of the SDLC.

In our series of eBooks, you'll find an explanation of why logging across the SDLC is essential in modern software delivery chains, as well as real-world examples of how teams can use Mezmo to streamline three distinct stages in the SDLC: Development, QA and staging, and production troubleshooting. This eBook is focused on debugging in development.

TABLE OF CONTENTS

DEBUGGING IN DEVELOPMENT WITH MEZMO

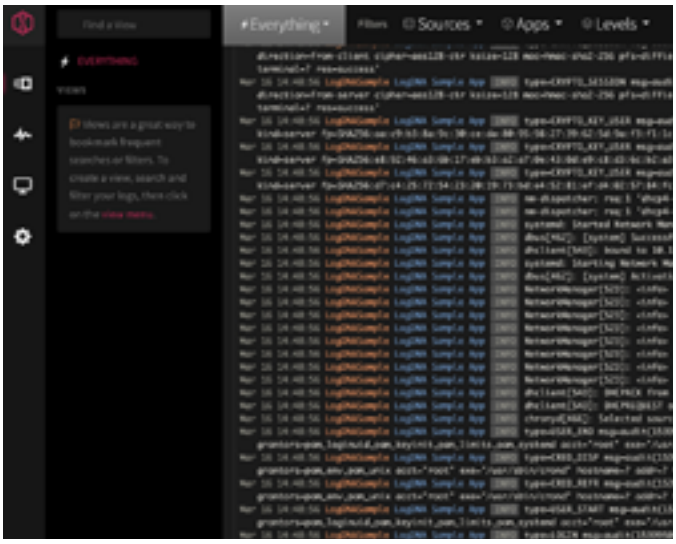
Developing scalable and reliable applications is a serious business. It requires precision, accuracy, effective teamwork, and convenient tooling. During the software construction phase, developers employ numerous techniques to debug and resolve issues within their programs. One of these techniques is to leverage monitoring and logging libraries to discover how the application behaves in edge cases or under load.

Centralized logging gives users access to the information that they need to effectively debug during the development process and Mezmo makes it easy to retain subsets of logs to meet different teams needs. For instance, developers often need access to a true depth of information from their logs, while SREs may be more interested in lightweight logging levels like info and trace. Read on to learn how the Mezmo platform empowers users at all levels of the development process.

Initial Development Environment Setup

The first thing you need to do is sign up with Mezmo. The process is very smooth. From here, you can explore their dashboard.

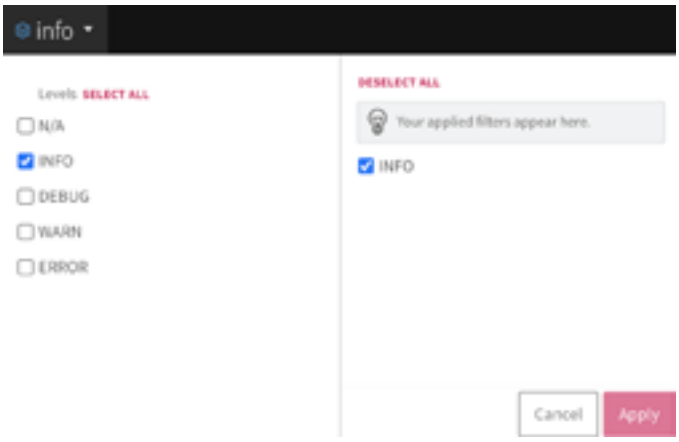
On the dashboard page, you have the option to pre-load sample log data or configure an agent collector yourself (or you can do both). If you select the sample data, you can add applications later. Here is what the screen looks like when the sample data is loaded:



All logs are clearly visible and itemized. When you select a log line, you can view all of the meta field information that was logged at that time. This is due to the automatic parsing of log lines as they are ingested into the Mezmo platform:

You also have the option to view in context. When you click this option, you can see a slice of the logs within the particular context of source, per app, or both. View in Context allows you to see the log lines that have lead up to this event as well as the lines that occured after the event:

You can also filter the logs by level. This is especially useful for eliminating most of the irrelevant noise when debugging. You can select the filter levels from the dropdown options at the top and apply them to the main view:



Next, we'll show you how to enroll a new application in the platform to test in development.

Enrolling a New Application

Mezmo supports ingestion from multiple sources using the Mezmo Agent, Syslog, Code Libraries, and APIs. In this example, we will enroll a Node.js application sourced from this repo.

You can follow the installation process as explained in the Readme. Then, you will need to hook the Mezmo logger into the Winston.js instance config.

```
$ npm install ip morgan mezmo-winston @
types/ip --save
```

Then modify the util/logger.ts file to include the Mezmo configuration:

```
import winston from "winston";

import mezmoWinston from "mezmo-winston";

import ip from "ip";

const mezmoOptions = {

key: "b5a09b29ad1d386964c61346108fc981",

     hostname: "localhost",

     ip: ip.address(),

     app: "Typescript-Node",

     env: "Production",

     indexMeta: true

};

const options: winston.LoggerOptions = {

     transports: [

        new winston.transports.Console({

        level: process.env.NODE_ENV ===
        "production" ? "error" :

"debug"

}),
        new winston.transports.

        File({filename: "debug.log", level:

"debug" })

    ],

};

const logger = winston.
createLogger(options);

options.handleExceptions = true;

logger.add(new
mezmoWinston(mezmoOptions));

try {

     throw new Error("It's a trap.");

} catch (err) {

     logger.error("Log from Mezmo-
     winston", {

         indexMeta: true

         , meta: {

             name: err.name | 'Error'

             , message: err.message

             , stack: err.stack

         }

     });

}

if (process.env.NODE_ENV !==
"production") {
    logger.debug("Logging initialized at
    debug level");

}
export default logger;
```

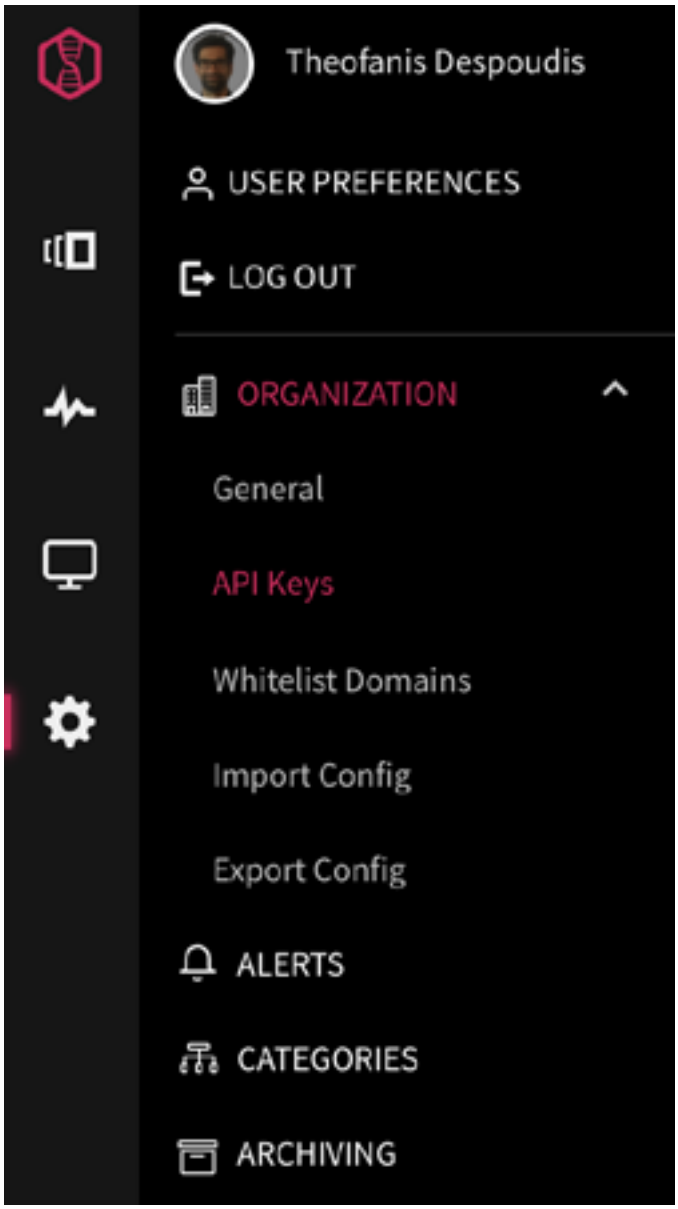
Then add an empty module definition for the

`mezmo-winston` package in

```
/src/types/mezmo-winston.d.ts
```

```
declare module 'mezmo-winston';
```

You will need to provide the secret API key for publishing logs in the mezmoOptions. This can be found in the Organization-> API Keys settings:

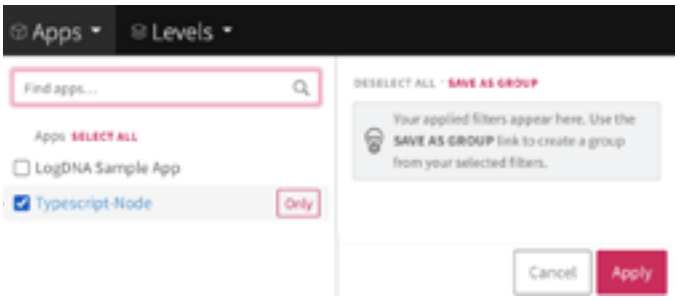


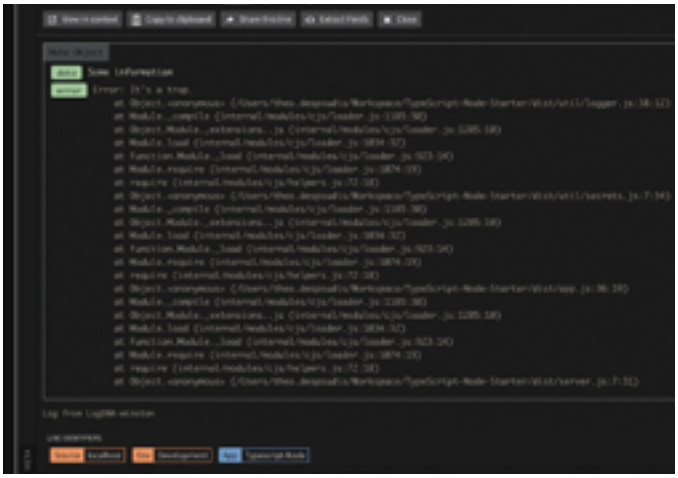
Once you have everything configured, you can start the development server and watch the dashboard as the new logs get populated:

```
$ npm run watch-debug
```

Navigate to localhost:3000 and make sure to enable live monitoring in the Mezmo platform. This can be found at the bottom right of the Mezmo dashboard.

Now you can see the new entries. If you are having trouble finding them, you may want to filter by application first and then select the application name.





Let's take a look at some of the other debugging utilities that Mezmo offers.

Debugging with Mezmo

Mezmo has several options and helpers for debugging applications. Let's explore them briefly one by one.

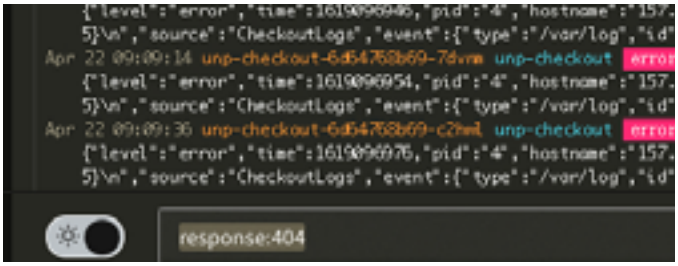
Tracebacks

If you check one of the logs after you have finished the logging configuration, you will be able to see tracebacks. That's because an error was thrown after the logger was configured and propagated into the platform. By lookingat the error trace, you can clearly see that the origin was in the `/dist/util/logger.js` file.

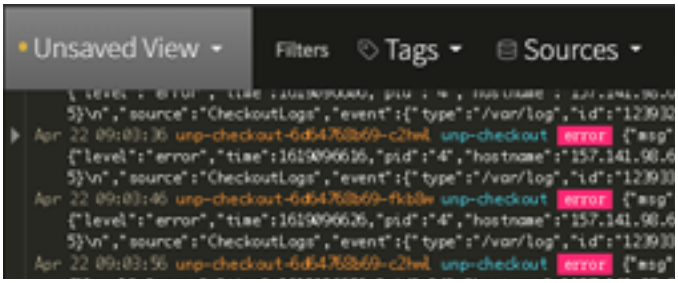
Alerts

Alerts are crucial to any technology as they give us a heads up when something is happening within our environment. With alerts we can get notifications through various means with Mezmo. Out of the box Mezmo supports alerts that can be triggered through email, PagerDuty, and Slack to name a few. Mezmo also supports webhooks for alerting capabilities.

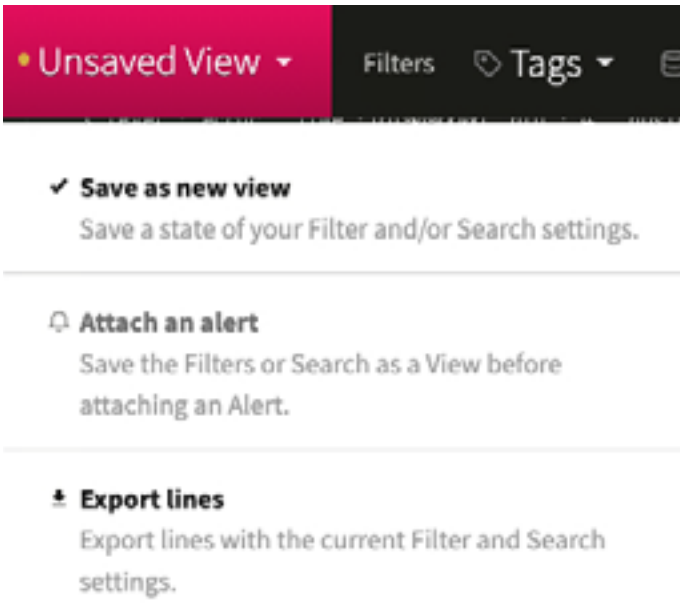
How do we set up an alert in Mezmo? Alerts start when we filter down our logs to a specific query we are interested in. Filtering can take place through several means within the platform, but for this example we will use the natural language query syntax to filter down 400 response errors that are typically specific to web applications.



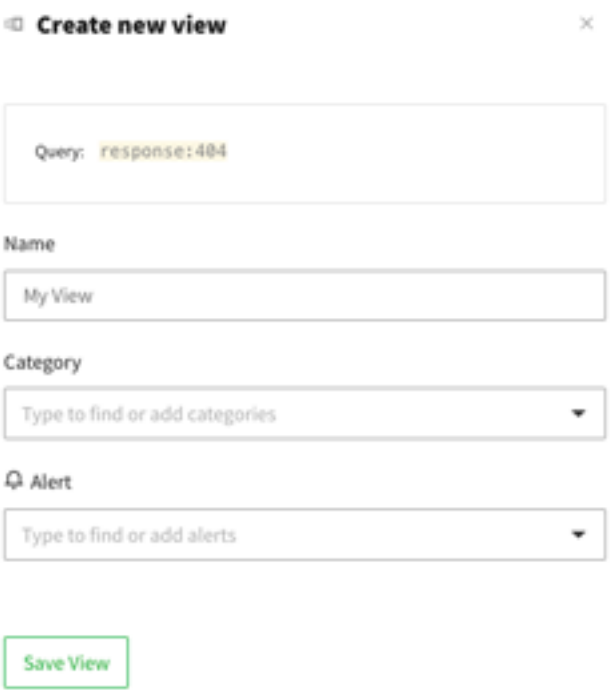
Now that we have our filter in place it's time to set up our alerting. For that you will notice that you will see your View change to 'Unsaved View' at the top of the Views page.



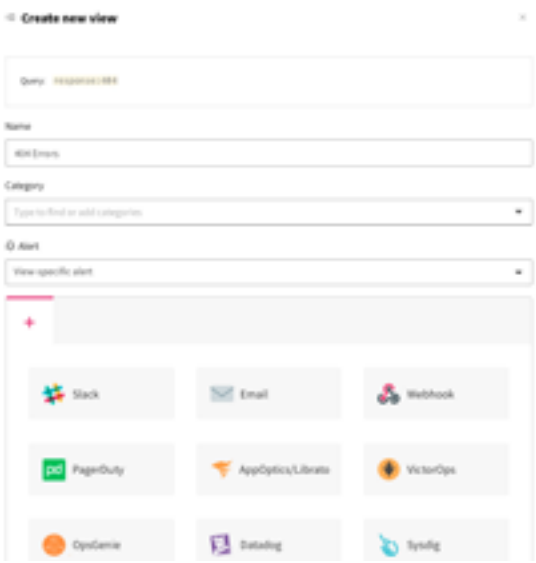
Clicking on the 'Unsaved View' will provide us options to save the View and attach an Alert to it. If you already had a saved View you would have the ability to attach an Alert to that existing View from this menu.



Let's look at what happens when we click on 'Save as new View'.

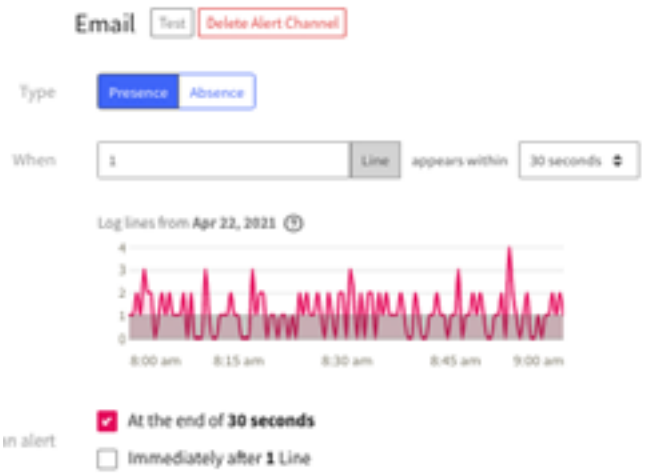


Within the pop up window we can give the View a name, add it to a category, and attach an Alert to it. When we go and attach an Alert to our View we are presented with the screen below.

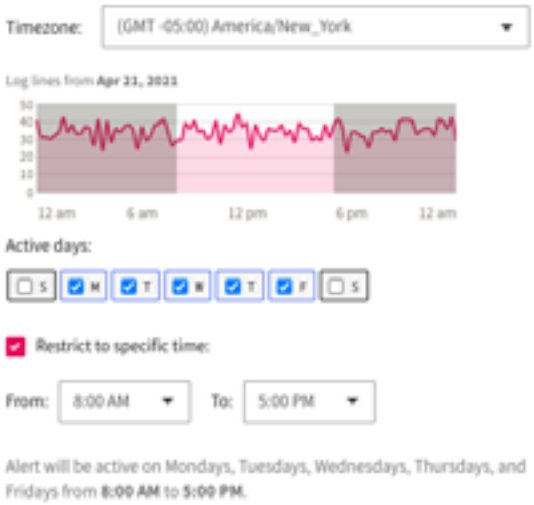


From here we can see the various alerting options that are provided out of the box, and also see the Webhook option as mentioned before. We will select email for our first option.

There is a lot to take in with the above screenshot so let's walk through it piece by piece.



The first section is all about alerting off either the presence or absence of log lines. Think of presence as meaning "when I see a defined number of lines come in during a specified period, I want to be alerted to this." Absence would be the opposite of that. It would mean "I'm expecting my application to generate X number of log lines and if it dips below that, then I want to be alerted as there may be issues with my application continuing to run and accept calls properly." We can also see when this Alert will be triggered based on our input with the gray line that runs across the display.



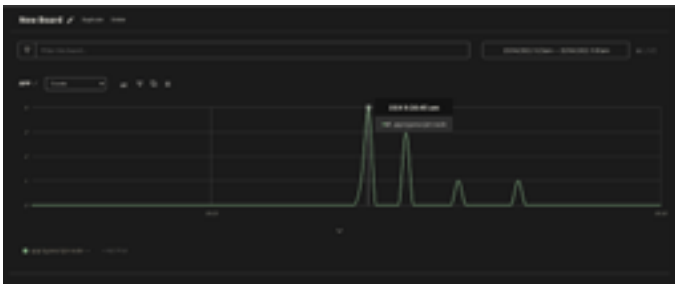
Next we can create custom schedules that define when this Alert is to be triggered. For this example we can specify typical working hours of Monday through Friday from 8:00 am to 5:00 pm. This is useful as we can create alert escalations that are sent to one place during normal operation hours and another place after hours or on the weekends.

Alerting is crucial these days with such busy schedules, remote working, and it helps avoid things like context switching where you'd have to be monitoring a web UI all the time.

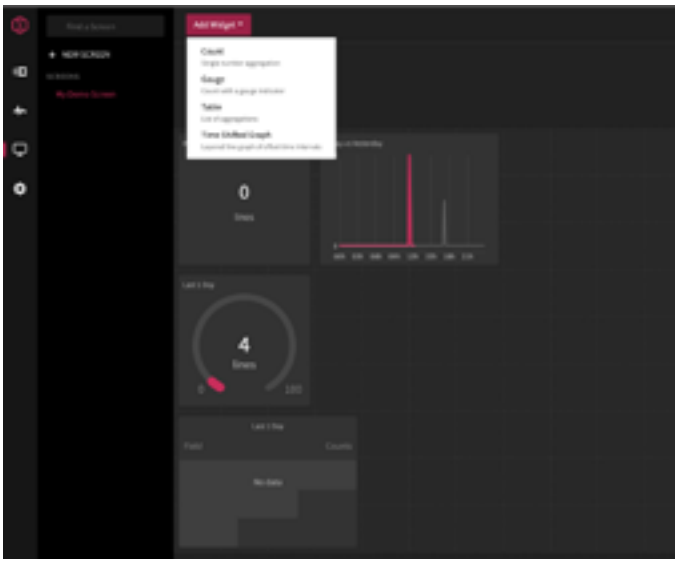
Boards and Screens

After setting up Alerts, you can create your own Board with custom widgets. For example, you can add a widget that uses only logs from a particular application:

In this example, if you select app and Typescript-Node, you will see the following graph:

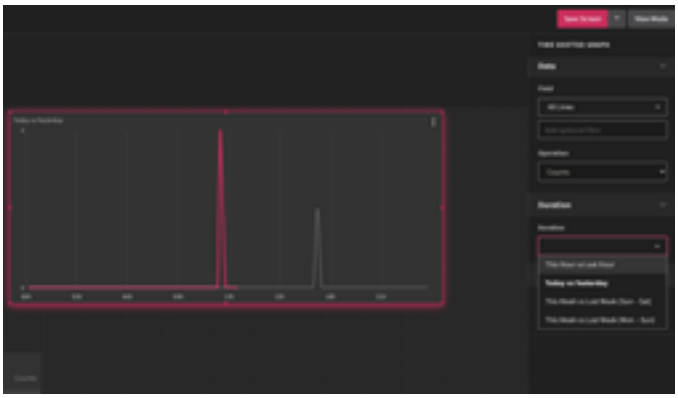


Screens are similar to Boards, but they give you a birds-eye view of your widgets. You can place them wherever you like.



Time-Shifted Graphs

After you've written some application logic, you can revisit the application in specific time intervals to check if the reliability has improved. This can be accomplished by using Time-shifted Graphs. With this feature, you can compare log events across two different time spans. To do so, you begin by selecting a widget from a screen. Then, using the sidebar options, you can change the duration field to provide valuable insights about the rate of events:



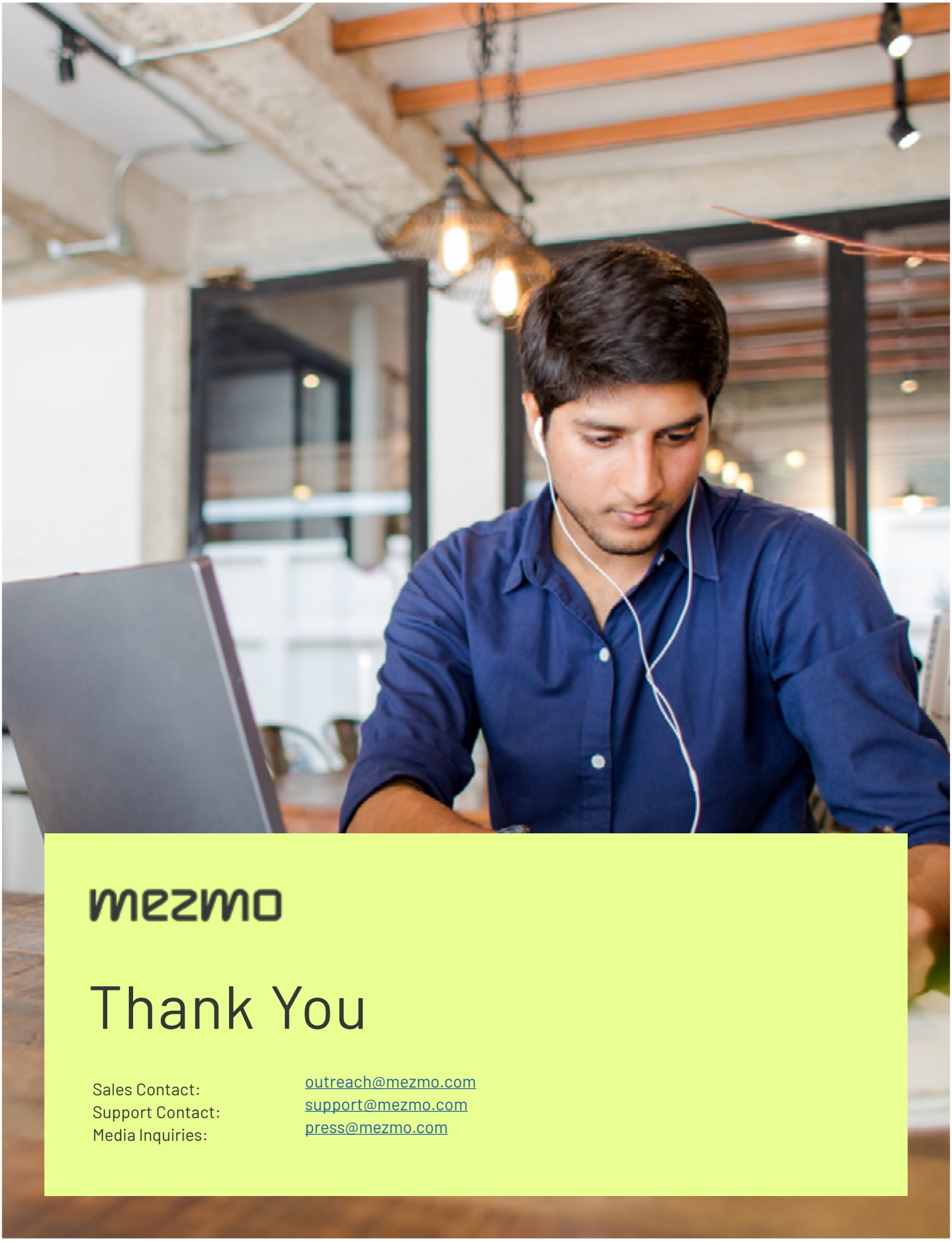
These Graphs are excellent for development, since they demonstrate the general tendency of the log events after new test cases have been written or major code changes have been implemented.

Next Steps with Mezmo

This eBook offered a brief tour of the main features of Mezmo's platform that cater to developers. We showed you how to review tracebacks, view in context, use Live Tail, and set up Mezmo Alerts for fundamental errors. Together with Boards, Graphs, and Screens, this platform gives developers a comprehensive set of tools for debugging applications. You can also take it to the next level by using Mezmo for production environments – but we'll explore that topic is another eBook in the series.

CONCLUSION

In this eBook, we've shown how to leverage logs and Mezmo to debug in development. Mezmo can help optimize other SDLC stages including QA and staging and production, which are discussed in the other eBooks in this series. No matter which stage of the SDLC you help manage, or which challenges you face, logs are one key resource to help you do your job better. And in a world where teams are expected to deliver new application releases multiple times per week, or even per day, engineers need every insight and data point available to them to keep the delivery pipeline flowing smoothly.



mezmo

Thank You

Sales Contact: outreach@mezmo.com
Support Contact: support@mezmo.com
Media Inquiries: press@mezmo.com