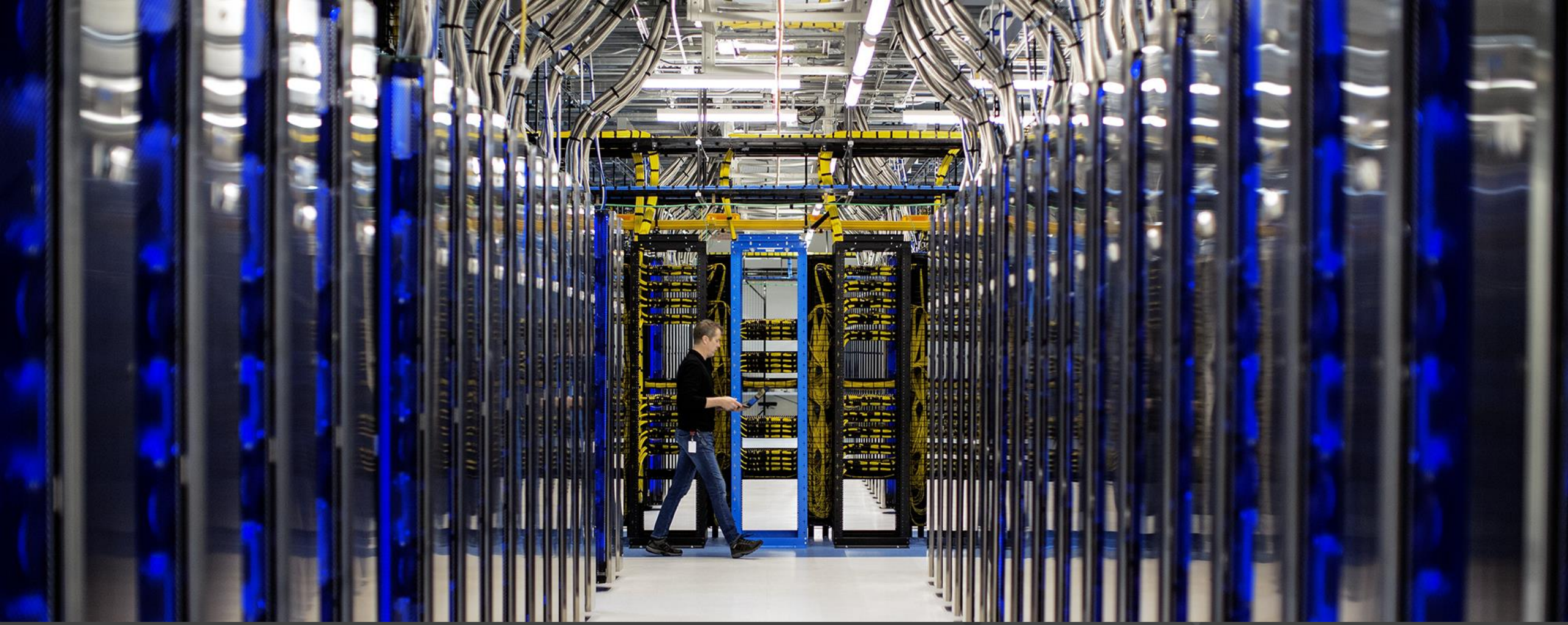
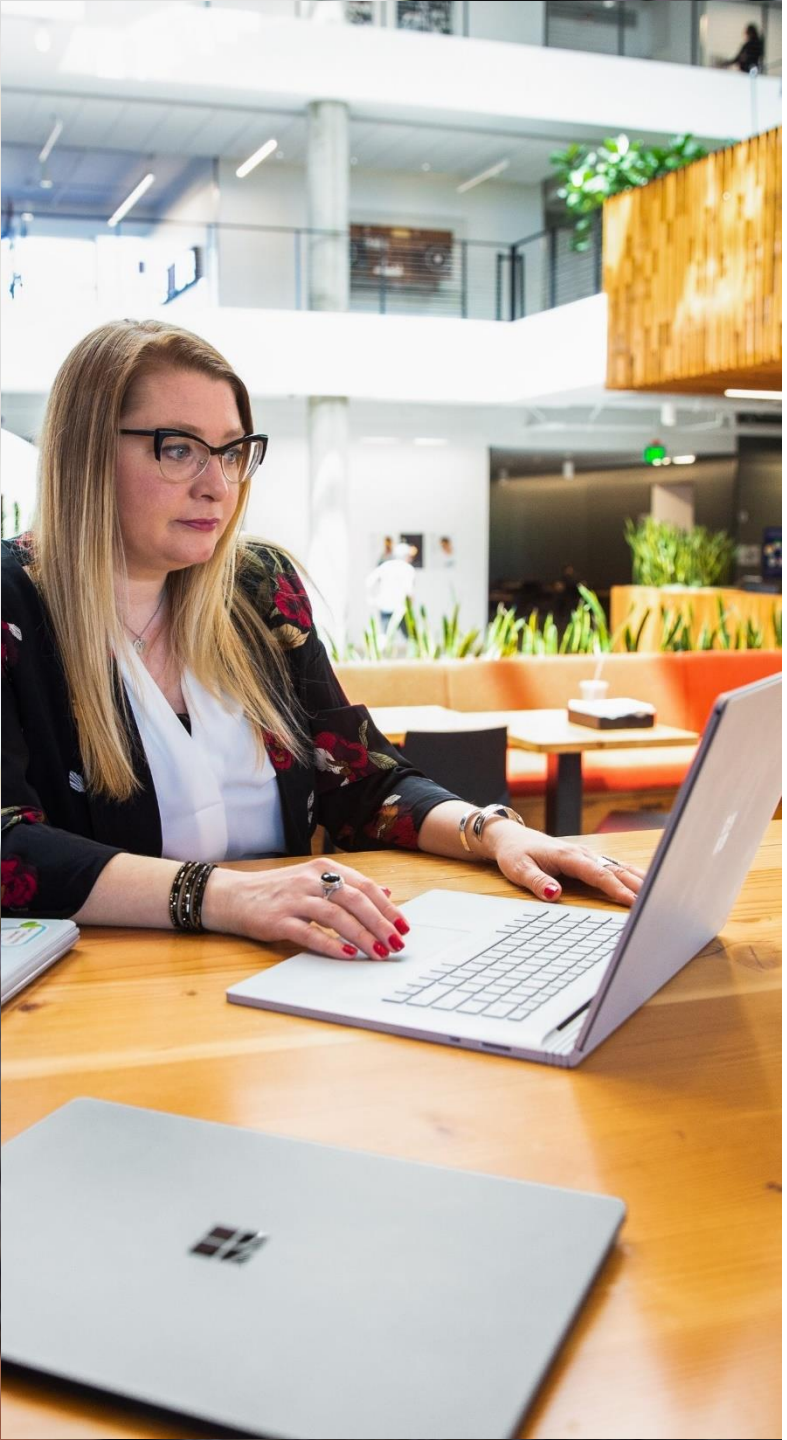


OSS | Specialist | Nodejs



Course Information

This course will include:

- Node.js on Linux Overview
- Deployment Scenarios
- Node Architecture & Memory Management
- Troubleshooting Performance Issues using V8 Profiler
- Installing and Using PM2
- How to Install and Configure Nginx
- Nodejs on Windows Configuration & Troubleshooting

Microsoft

Node.js on Linux

Module Objectives

- Use different tools to **create** and **configure** Node.js applications for *App Service Linux*.
- Manage **issues** related to *Deployment*, *High Memory* and *High CPU* for Node.js applications running on *App Service Linux*.
- Explore the process to **install** and **connect** *Relational* and *NoSQL Databases* to Node.js applications.
- Compare different options to **enable**, **access** and **review** *Application Logs*.

- Open your Participant Guide.
- Review the information provided.
- Follow Instructions to install and set up Azure CLI, Visual Code and Node.js.

Hands On: Install Azure CLI, Visual Code and Node.js

Create and Deploy Node.js Apps

Node.js Apps can be created by using the following tools: **Azure Portal**, **Visual Code**, **Azure CLI**, or **ARM Templates**.

There are several **deployment options** for *Node.js* on *Azure Web App Linux*:

Type	Description
CD (Kudu Build Service or Azure Pipelines)	GitHub, BitBucket, and Azure Repos repositories by pulling in the latest updates.
Local Git (Kudu Build Service or Azure Pipelines)	Git repository on your local computer.
ZipDeploy (Kudu Service)	ZIP File deployment through Kudu.
Run from Package (Kudu Service)	ZIP Package File deployment as read-only.
FTP	Manual Deployment to copy files through FTP and FTPS endpoints.
GitHub Actions (Preview)	Automated software development lifecycle workflow in GitHub.

Foundational Knowledge

Type	Description
Node JS	• Node.JS is an open-source, cross-platform, JavaScript environment that executes JavaScript code outside of your typical browser. • Node.JS is asynchronous (non-blocking) • JavaScript is a programming language, Node.JS is not. • Node.JS is a JavaScript program.
NPM/Yarn	• NPM stands for Node Package Manager • NPM is a package manager for the JavaScript programming language • NPM is the default package manager for Node.JS • Yarn is similar to NPM but provides some of performance and security improvements over NPM
NPX	• NPX is a tool intended to help round out the experience of using packages from the NPM registry. • The same way NPM makes it super easy to install and manage dependencies hosted on the registry. • NPX makes it easy to use CLI tools and other executables hosted on the registry.
Package.json file	• Used by NPM and Yarn. • Lists the packages your project depends on • Specifies versions of a package that your project can use using semantic versioning rules • Makes your build reproducible, and therefore easier to share with other developers
Build/Runtime	• The Oryx system comprises *Build* and *Run* images. **Build** images include compilers, libraries, headers and other tools necessary to prepare artifacts; **Run** images are smaller and contain only components required to run programs. Build Image is Kudu & Runtime image is the application container.

Oryx for Node.js

Oryx NodeJS Detection

It runs in **KuduLite** when the following conditions are met:

1. One of these files is found in the root of the repo: `package.json`, `package-lock.json`, or `yarn.lock`.
2. One of these files is found in the root of the repo: `server.js`, or `app.js`.

Oryx Build

It runs in **KuduLite**:

1. Run custom script if specified by **`PRE_BUILD_SCRIPT_PATH`**.
2. Run **`npm install`** without flags.
3. Run **`npm run build`** if a build script is specified in the *package.json*.
4. Run **`npm run build:azure`** if a build:azure script is specified in your *package.json*.
5. Run custom script if specified by **`POST_BUILD_SCRIPT_PATH`**.

Run and Startup for NodeJS

It runs in a **Node.js container**:

1. Run `npm start` if a start script is specified.
2. Else, if a script is specified in `package.json` main field, run that.
3. Run the first script found in the root of the repo, which can be: `bin/www`, `server.js`, `app.js`, `index.js`, or `hostingstart.js`.

Node Versions

To configure & review Node version information, you can use Azure Portal, Azure CLI.

Display & Set Node Version

1. Go to **Cloud Shell**, and run the following commands:

Command	Description
`az webapp config show --resource-group <resource-group-name> --name <app-name> --query linuxFxVersion`	Display the current Runtime version.
`az webapp list-runtimes --linux | grep NODE`	Display all supported Node versions
`az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "NODE|10.14”`	Set Node Version

Runtime Support & Patching

- **Depreciations will be announced & once a version reaches end of life, it will no longer be available as an option for new deployments. Existing apps will show as "Depreciated" from the Portal dropdown.**

Version	Status	End
Node 9.x	EOL	6/30/19
Node 10.x	EOL	4/30/21
Node 12.x	LTS	4/30/22
Node 14.x	LTS	4/330/23

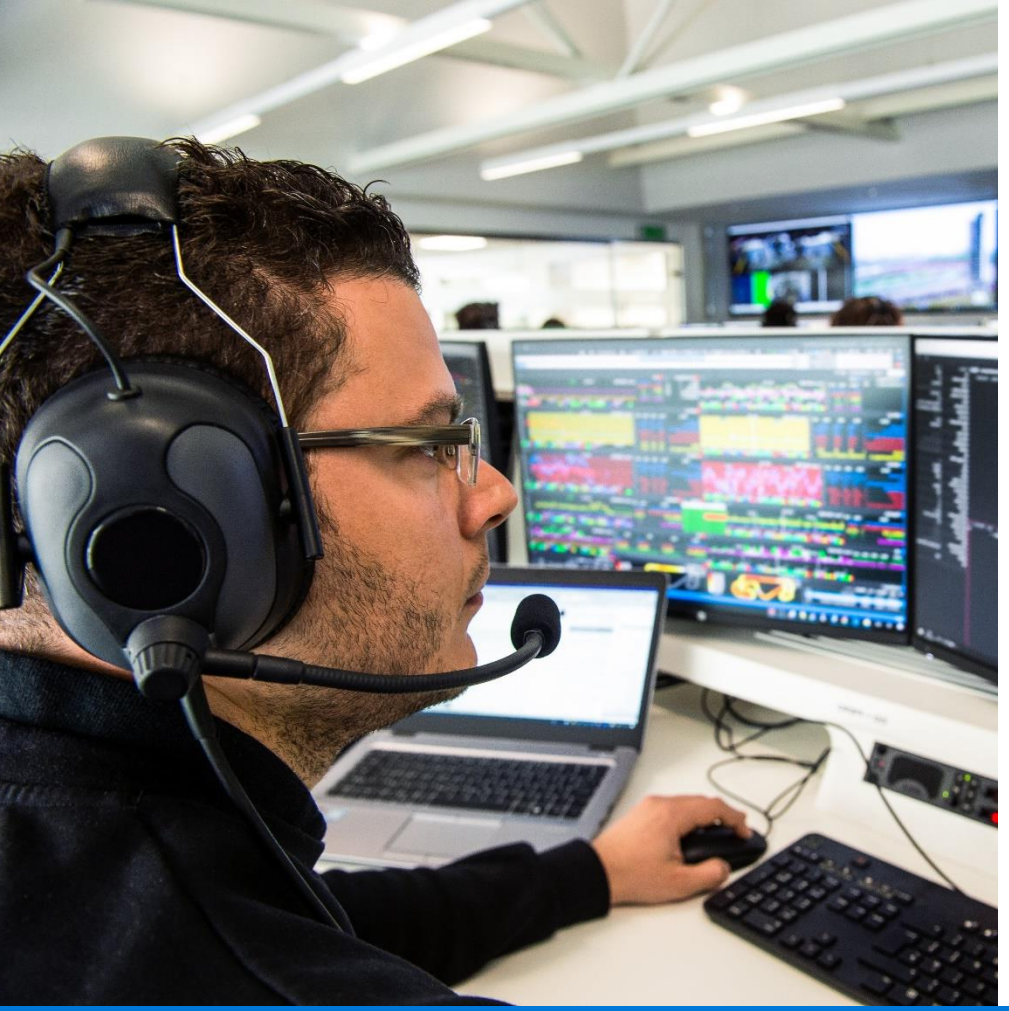
CD (Kudu Build Service or Azure Pipelines)

You can enable CD from **GitHub**, **Bitbucket** and **Azure Repos**:

1. Ensure that your **repository root** has the correct files in your project with a **start script**.
2. Authorize **Azure App Service** to connect with the repository.
3. Select one of the two build servers:
    a. **Kudu App Service**: KuduLite uses Oryx as a build system to detect, install and deploy the app.
    b. **Azure Pipelines**: DevOps uses a pipeline with agent to build and deploy the app.

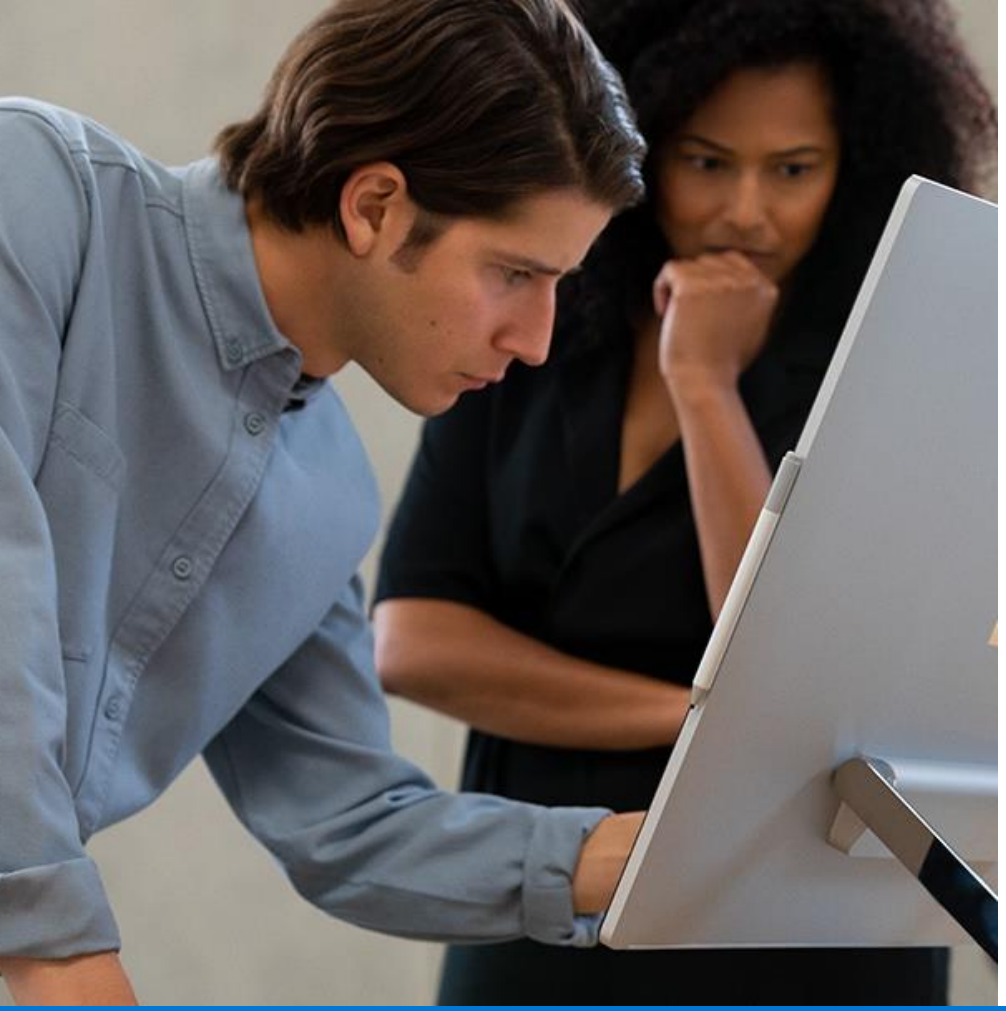
- Open your Participant Guide.
- Review the information provided.
- Follow instructions to deploy a Node.js App to Azure Web App from a forked repository on GitHub.

Hands On: GitHub with Oryx (Kudu Build Service)



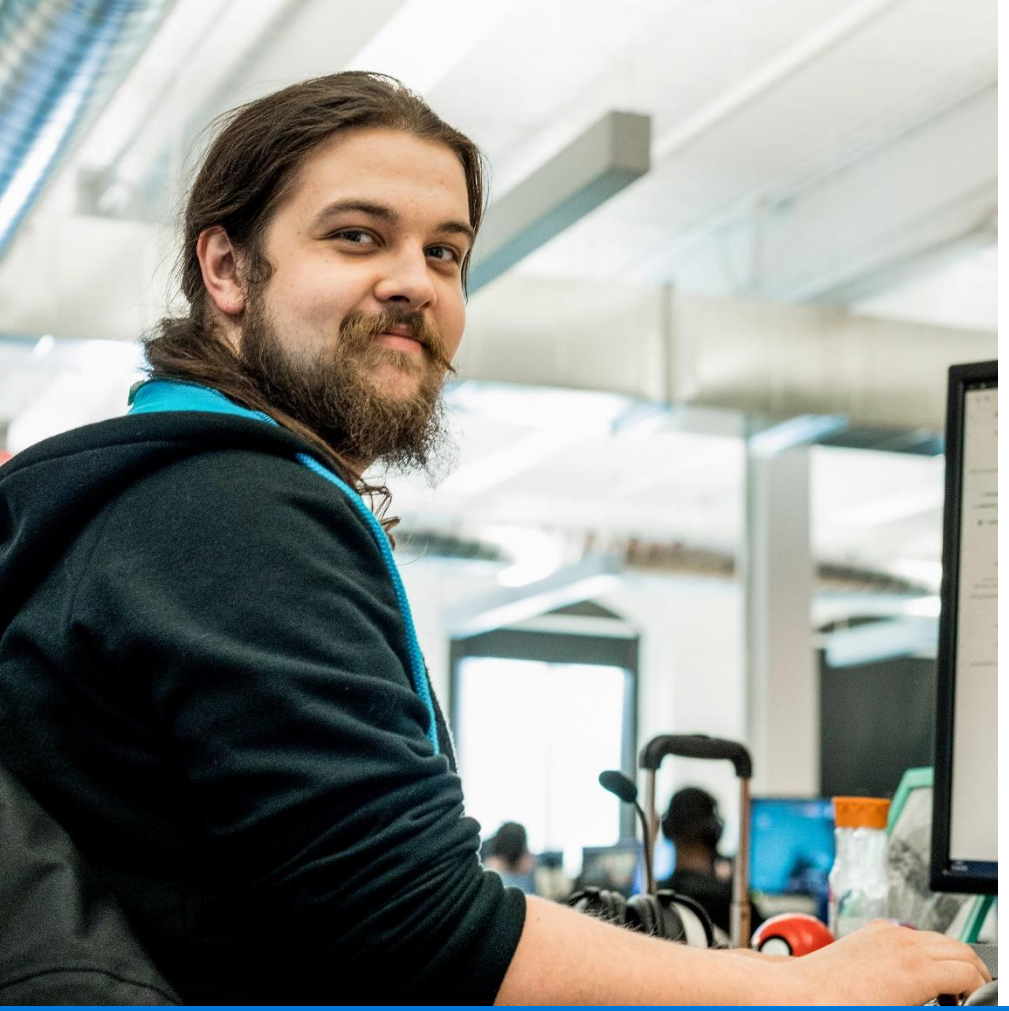
- Open your Participant Guide.
- Review the information provided.
- Follow instructions to create a site and deploy it using local git using Azure Portal and Git Commands.

Hands On: Local Git with Oryx (Kudu Build Service)



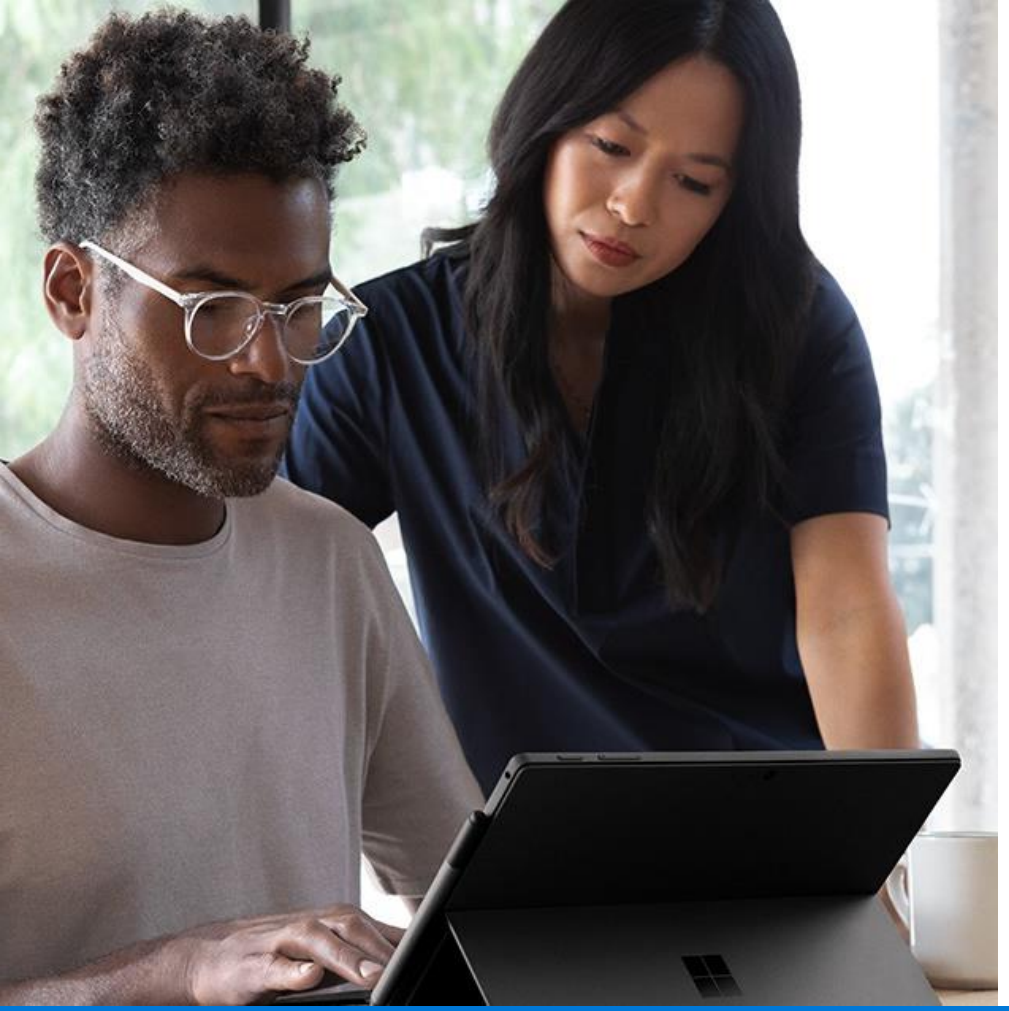
- Open your Participant Guide.
- Review the information provided.
- Follow instructions to deploy a Node.js application without the node_modules included and have Oryx build the packages for you.

Hands On: ZipDeploy with Oryx



- Open your Participant Guide.
- Review the information provided.
- Follow instructions to configure and deploy a Node.js application that Run From Package.

Hands On: Configure and Deploy Node.js Apps



- Open your Participant Guide.
- Review the information provided.
- Follow instructions to deploy Next with TypeScript app, and how to troubleshoot issues related to this deployment.

Hands On: Deploy and Troubleshoot Next with TypeScript app

Memory Management Components

The **main components** are the following:

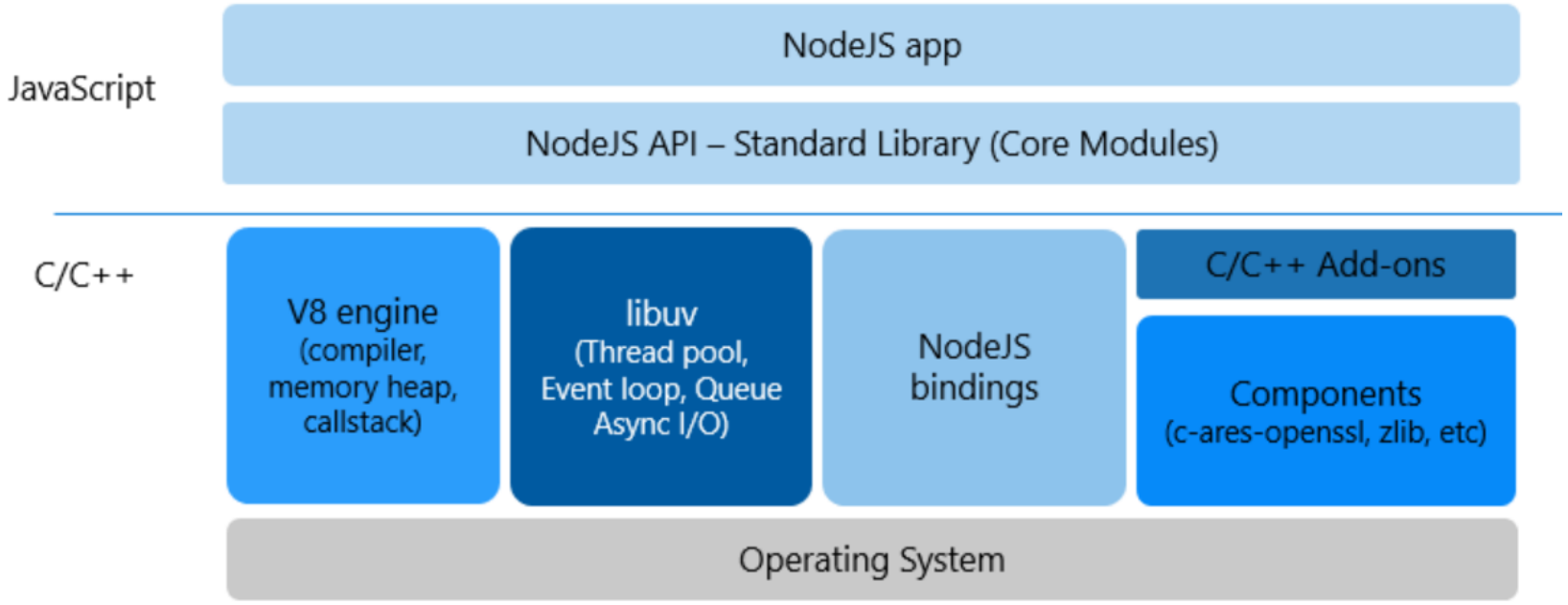
- Stack
- Heap
- Resident set size (RSS)
- Working set size (WSS)
- Virtual memory
- Virtual set size
- Garbage collector (GC)

Node.js Architecture

Node.js handles concurrent requests with **Single-Threaded model** and it is built on **Chrome's V8 JavaScript engine**.

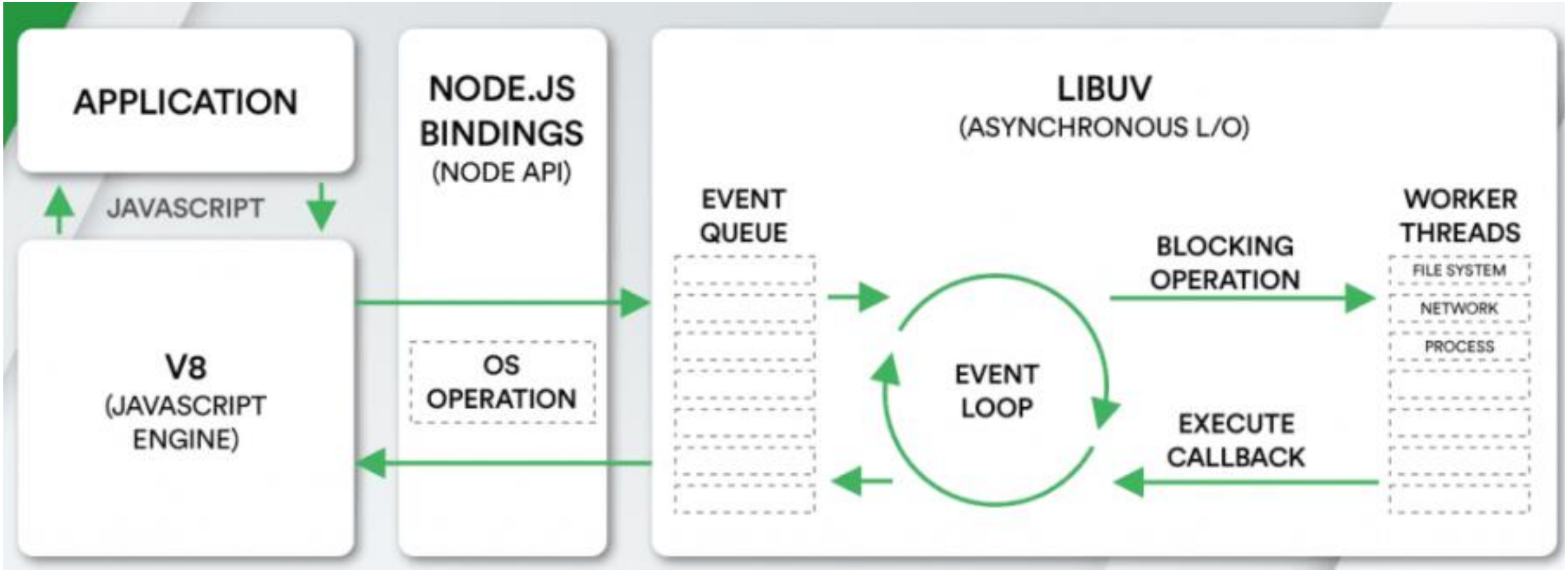
It is integrated by:

- Node.js App
- Node.js API
- Node.js Standard Library
- V8 JavaScript Engine
- LibUv
- Node.js Bindings
- Components



Node.js Architecture

Node.js handles concurrent requests with **Single-Threaded model** and it is built on **Chrome's V8 JavaScript engine**.

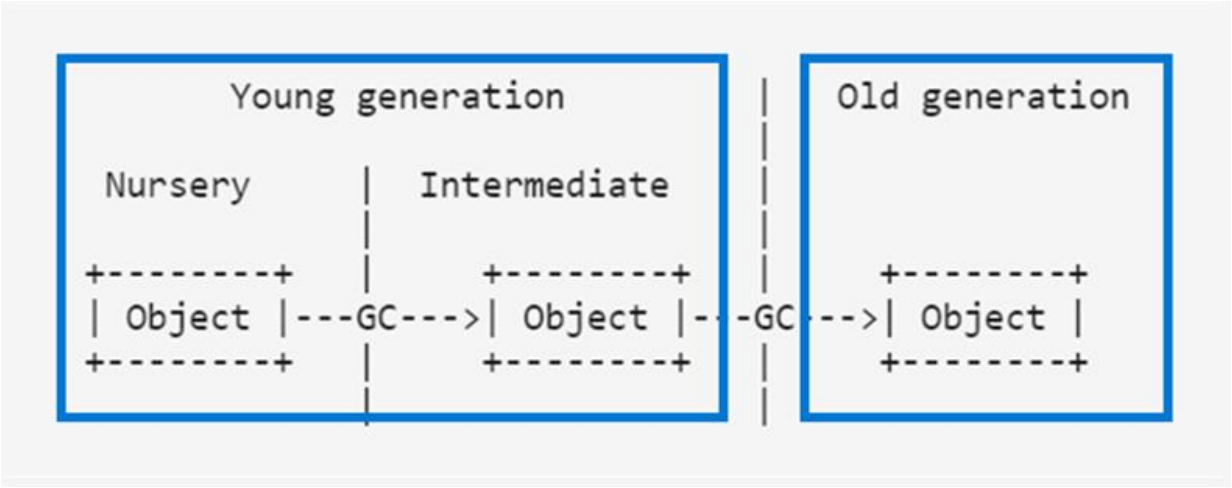


V8 Generational Heap Layout

Generations

There are two generations:

- Young generation (or New Space)
- Old generation (or Old Space)



Ages of objects

There are three ages of objects:

- Nursery age
- Intermediate age
- Old generation

Garbage collectors

There are two generations:

- Minor GC (Scavenger)
- Major GC (Mark-Compact)

V8 GC Flow Between Spaces

Young Generation

It is divided into **To-Space** and **From-Space**. This space is managed by the **Minor GC (Scavenger)** process which consists in:

- Copy the object from the **Nursery** area called **From-Space** to the **Intermediate** area called **To-Space** after one GC.
- In the second GC, the objects that were not cleaned in the Young Generation are moved to the **Old Generation** area called **Old-Space**.

Old Generation

It is divided into **Old-Pointer Space** and **Old-Data Space**. This space is managed by the **Major GC (Mark-Sweep & Mark-Compact)** process which consists in:

- **Marking:** In this process reachable objects are found, checking from the known object's pointers. GC will follow each pointer to the object and marks it as reachable.
- **Sweeping**: It is the process where the memory addresses of objects are not marked alive. These objects are added into a data structure called a free-list.
- **Compaction**: It is the process of moving all survived objects into other pages using the free-list and compact those.

V8 Heap and Stack

The size of heap for V8 is **hard limit** and can **vary** between Node.js versions and architecture:

Version	Architecture	Size
Node.js 12/14	64 bits	2 GB
Node.js 6, 9 and 10	64 bits	1.5 GB
Node.js 12/14	32 bits	1 GB

The default size of stack region v8 allowed to use is **984 KB**.

Troubleshooting High Memory and CPU

1. Detect High Memory or High CPU issues with CSS internal tools.
2. Use Memory or CPU Profilers.

Reading Profile Traces

To find Memory and Profiler Tabs, follow the next steps:

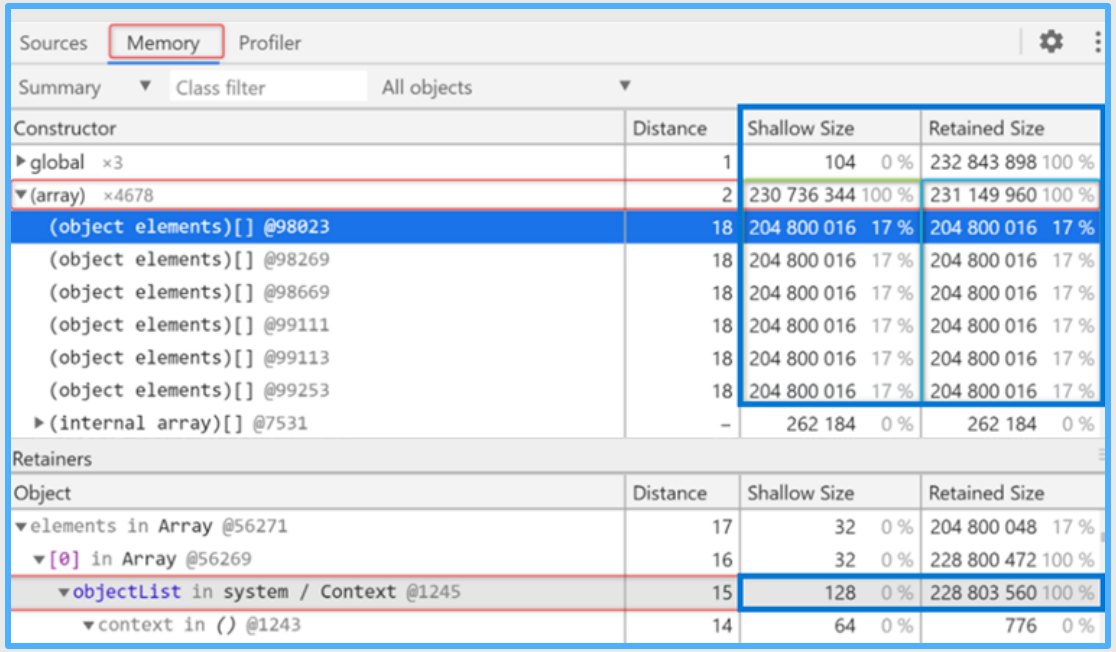
1. Navigate to *Chrome Browser*.
2. Type **chrome://inspect/.**
3. Press in *Open* dedicated *DevTools* for Node.

Memory / CPU Profiles

Memory Profiles

It allows to analyze:

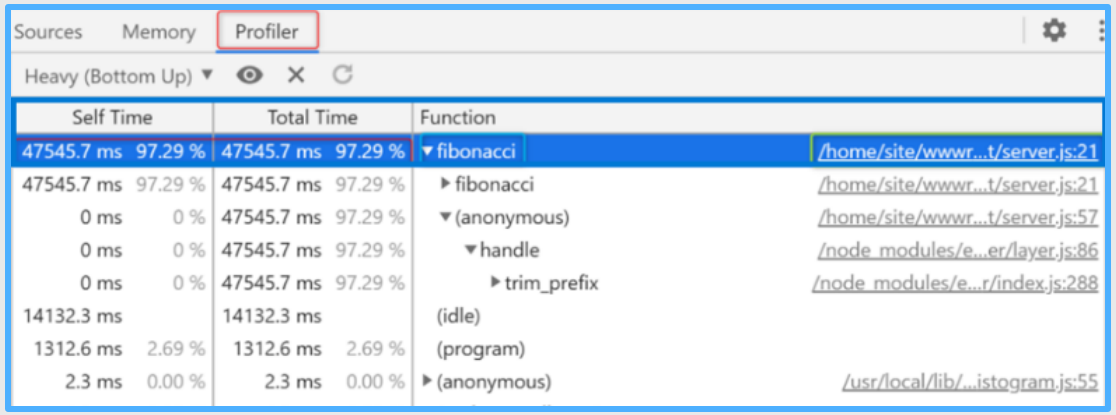
- **Shallow Size**: Size of memory that is held by the object itself.
- **Retained Size**: Size of memory that is freed once the object is deleted.



CPU Profiles

It allows to analyze:

- **Self Time**: Time spent in the function at the current level of a call tree.
- **Total Time**: Self time plus the amount of time it took to execute the code in functions that the current level calls.



- Open your Participant Guide.
- Review the information provided.
- Follow instructions to profile Memory and CPU using v8 Profiler.

Hands On: Profiling Samples

Connect Databases

To install a database driver for Node.js:

1. Use the command **`npm install driver@version`**.
2. Import the module in node as followed **`var driver = require('driver');`**.

Kind	Type	Driver	Installation
Relational	MS SQL Server	mssql (Tedious/msnodesqlv8)	npm install mssql
Relational	MySQL	mysql	npm install mysql
Relational	MariaDB	mariadb	npm install mariadb
Relational	Oracle	oracledb	npm install oracledb
Relational	PostgreSQL	pg	npm install pg
Relational	SQLite	node-sqlite3	npm install node-sqlite
NoSQL	MongoDB	mongodb	npm install mongodb
NoSQL	CosmosDB	cosmos	npm install @azure/cosmos
NoSQL	Cassandra	cassandra-driver	npm install cassandra-driver
NoSQL	Redis	redis	npm install redis
NoSQL	Apache CouchDB	nano	npm install nano
NoSQL	RethinkDB	rethinkdb	npm install rethinkdb
NoSQL	OrientDB	orientjs	npm install orientjs
NoSQL	MarkLogic	marklogic	npm install marklogic
NoSQL	ArangoDB	arangojs	npm install arangojs@5

- Open your Participant Guide.
- Review the information provided.
- Follow instructions to run a PostgreSQL sample.

Hands On: Running a PostgreSQL Sample

Application Logging

Application Logging can be enabled from the next tools:

From Azure Portal

1. Navigate to your app and select **App Service Logs**.
2. In Application logging, select **File System**.
3. In **Quota (MB)**, specify the disk quota for the application logs.
4. In **Retention Period (Days)**, set the number of days the logs should be retained.
5. When finished, select **Save**.

From Azure CLI

1. Install the **Azure CLI**.
2. Run the command **`az login`**.
3. Run the command **`az webapp log config --name <app-name> --resource-group <yourResourceGroup> --docker-container-logging filesystem`**.

Access Logs

Customer Side

- **Azure Portal**: Use **Log Stream** blade.
- **Kudu Site**: Download ZIP file: https://<app-name>.scm.azurewebsites.net/api/logs/docker/zip. Then, browse the logs: https://<app-name>.scm.azurewebsites.net/api/logs/docker, copy the link for any specific *Dockerfile* and open it in a new tab. Use *Kudu Bash/SSH* and navigate through **/home/LogFiles**.
- **Azure CLI**: Run the command `az webapp log tail --name <app-name> --resource-group <yourResourceGroup>`.

CSS Side

You can use **AppLens** and navigate to **Application Logs Detector**.

Read Logs

The **Logfiles directory** contains a list of files with logs displayed in the following format: `year_month_day_roleinstance_default_docker.log (ie. 2020_05_28_RD0123456789AB_default_docker.log)`.

`_default_docker.log` refers to your Application Logs, while `_docker.log` refers to the overall Docker Log.

Summary Lesson

Nodej.s App can be created with **Azure Portal**, **Visual Code**, **Azure CLI**, or **ARM Templates**.

The phases to **deploy a Node.js App** are Oryx Node Detection, Oryx Build, and Run and Startup for Node.js.

You can enable CD from **GitHub**, **Bitbucket** and **Azure Repos** repositories.

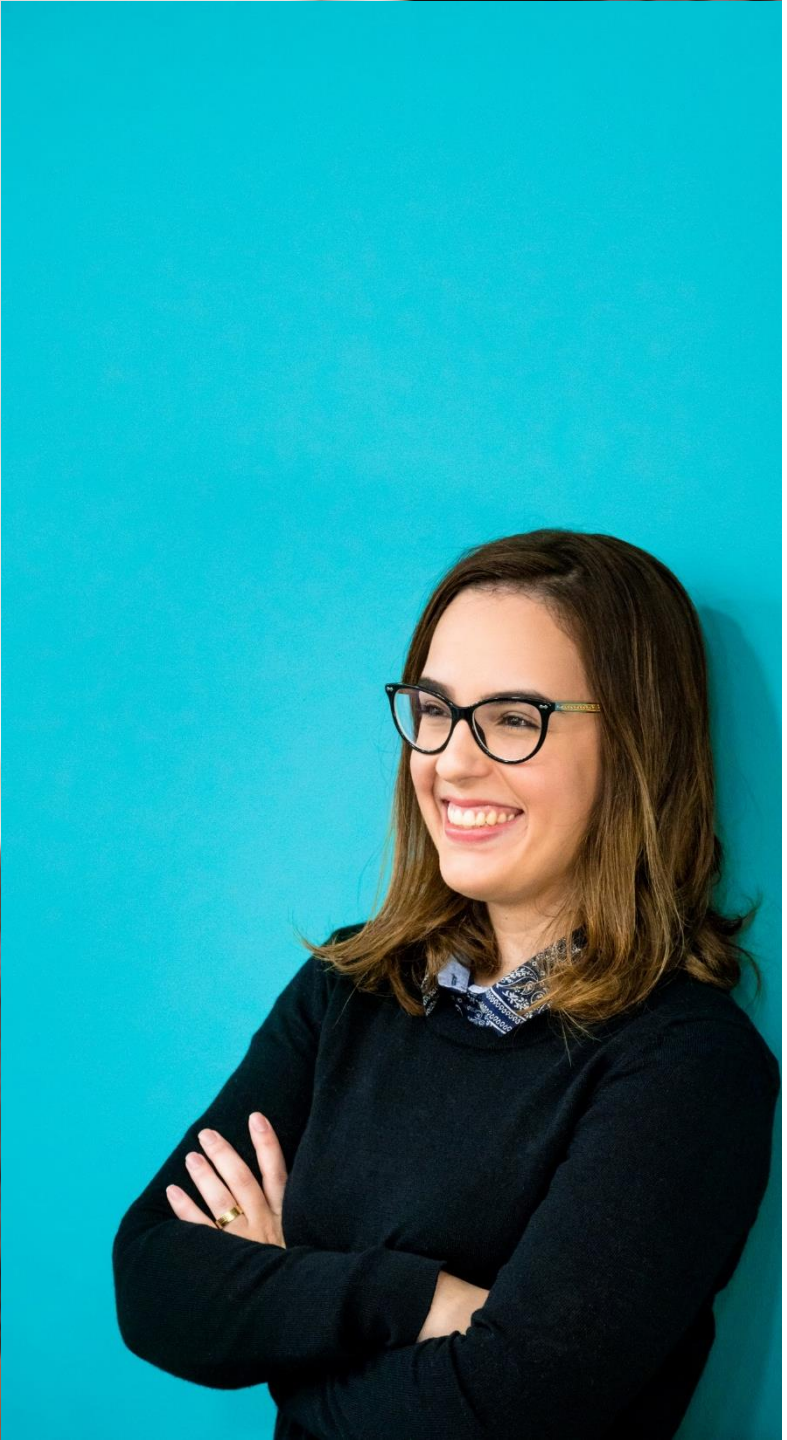
The **Memory Management Components** are stack, heap, RSS, WSS, virtual memory, virtual set size, and GC.

To troubleshoot **Memory and CPU issues**, you must detect them with CSS internal tools, and then use Memory or CPU profilers.

Node.js supports all kinds of databases, including **Relational** or **NoSQL Databases**.

Microsoft

Installing and Using PM2



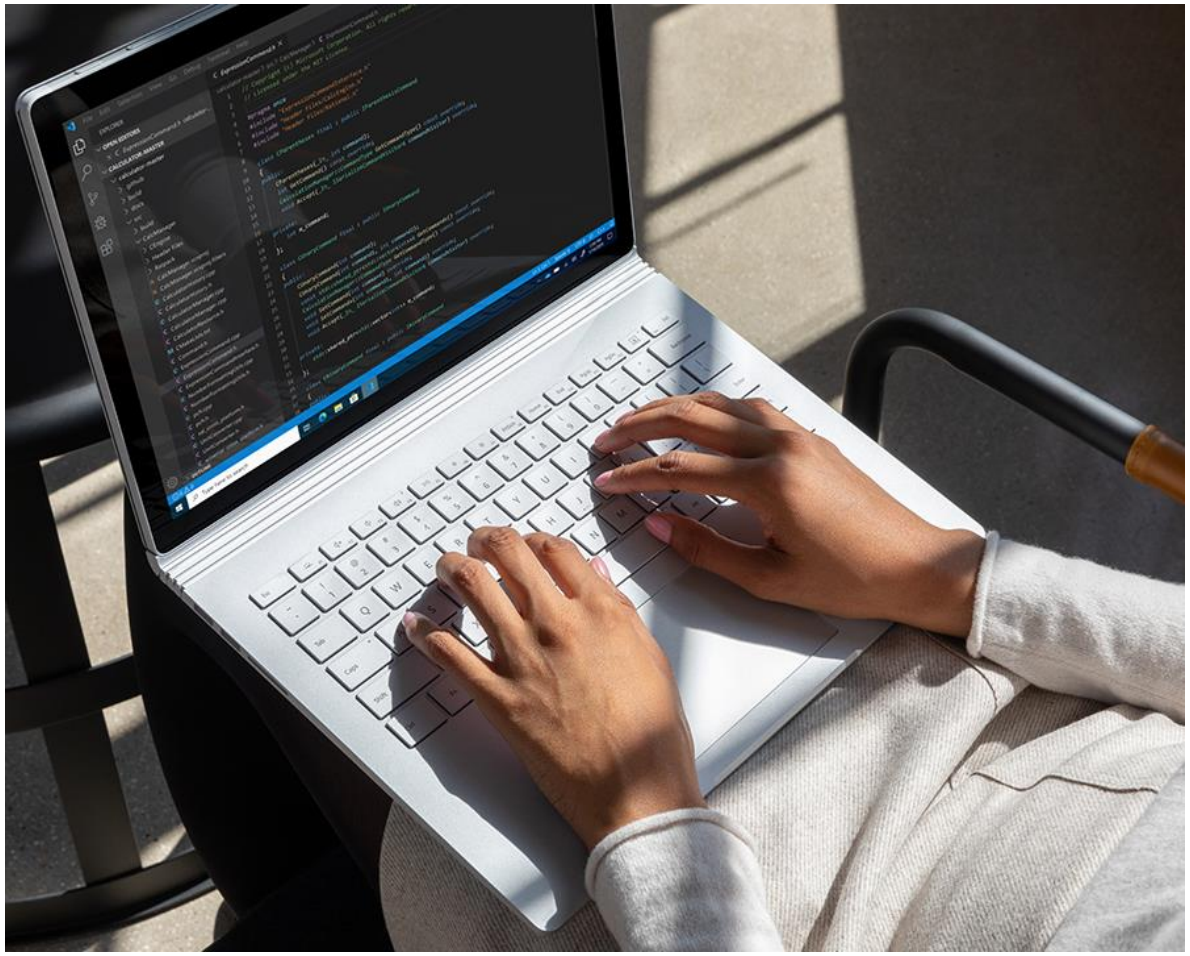
Module Objectives

- Understand how to install and configure PM2.
- How to monitor and review logs with apps running under PM2.
- How to configure PM2 Azure Web Apps Linux and serve static files.

Overview

PM2 is an advanced, production process manager for Node.js, it is known as daemon process manager that helps you manage and keep your application online.

It is a built-in Load Balancer that implements auto-restarting across crashes and machine restarts.



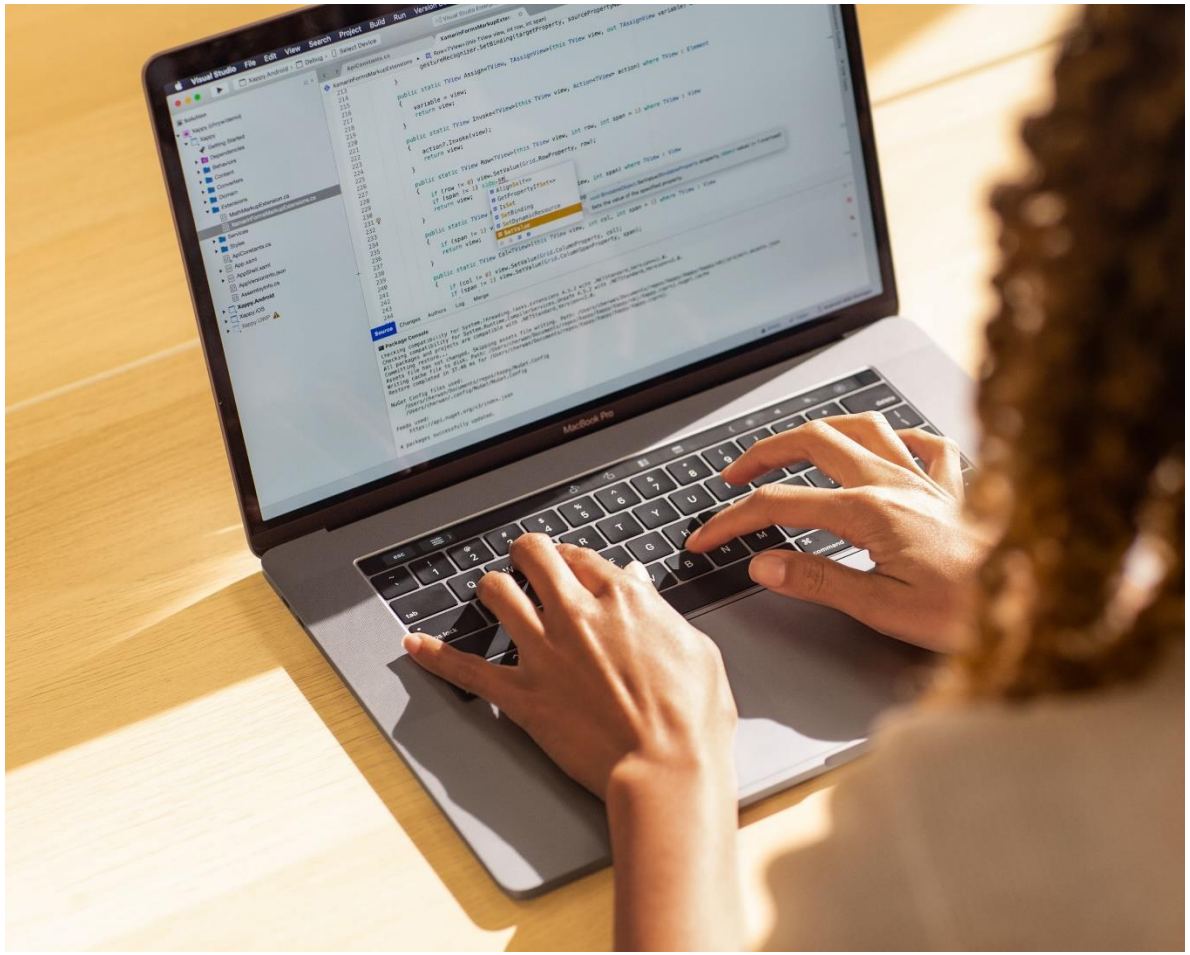
Installation

You can install pm2 globally with the following command:

```
npm install pm2 -g
```

Or using Yarn:

```
yarn global add pm2
```



Configuration and Startup

To start a nodejs application you can use the following command:

```
pm2 start server.js
```

To start a nodejs application passing node arguments, you can use the following command:

```
pm2 start server.js --node-args="--max_old_space_size=2048"
```

You can use other options to pass through the CLI as followed:

```
--watch Watch and Restart app when files change.

--max-memory-restart <200MB>Set memory threshold for app reload

--log <log_path> Specify log file

--no-daemon Use to run in docker container to prevent exit.
```

You can use a configuration file called Ecosystem File (ecosystem.config.js) or with yaml file (<name>.yml) to set specific parameters:

```
pm2 start ecosystem.config.js
```

Configuration and Startup Continued

Using yml file:

```
pm2 start process.yml
```

To the right you'll see a quick example of an ecosystem file.

```
module.exports = {
  apps : [
      {
        name: "myapp1",
        script: "./server.js",
        watch: false,
        error_file:'./error.log',
        out_file:'./output.log',
        env: {
          "NODE_ENV": "development",
          "PORT": 3000,
        },
        env_production : {
          "NODE_ENV": "production",
          "PORT": 8080,
        }
      }
  ]
}
```

Monitoring and Logs

To list the status of all applications managed by PM2, you can use the following command:

```
pm2 [list|ls|status]
```

To review logs from PM2, it will tail the last 15 lines for all process, but you can also modify this parameter, you can use the following command:

```
pm2 logs <app_name|all>
```

To have a realtime dashboard for checking CPU, Memory, Crashes, etc., you can use the following command:

```
pm2 monit
```

```
edison@edisgavm:$ pm2 logs
[TAILING] Tailing last 15 lines for [all] processes (change the value with --lines option)
/home/edison/.pm2/pm2.log last 15 lines:
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] starting in -fork mode-
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] online
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] exited with code [1] via signal [S
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] starting in -fork mode-
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] online
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] exited with code [1] via signal [S
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] starting in -fork mode-
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] online
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] exited with code [1] via signal [S
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] starting in -fork mode-
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] online
PM2        | 2020-05-31T17:43:14: PM2 log: App [server:0] exited with code [1] via signal [S
PM2        | 2020-05-31T17:43:14: PM2 log: Script /home/edison/codes/nodejs/nodejs-helloworl
PM2        | 2020-05-31T17:43:51: PM2 log: Stopping app:server id:0
PM2        | 2020-05-31T17:43:51: PM2 error: app=server id=0 does not have a pid

/home/edison/.pm2/logs/server-out.log last 15 lines:
/home/edison/.pm2/logs/server-error.log last 15 lines:
0|server   |     at Function.Module._resolveFilename (internal/modules/cjs/loader.js:625:15)
0|server   |     at Module.Hook._require.Module.require (/home/edison/.nvm/versions/node/v12
0|server   |     at require (internal/modules/cjs/helpers.js:16:16)
0|server   |     at Object.<anonymous> (/home/edison/codes/nodejs/nodejs-helloworld-processv
0|server   |     at Module._compile (internal/modules/cjs/loader.js:776:30)
0|server   |     at Object.Module._extensions..js (internal/modules/cjs/loader.js:787:10)
0|server   |     at Module.load (internal/modules/cjs/loader.js:643:32)
0|server   |     at Function.Module._load (internal/modules/cjs/loader.js:556:12)
0|server   |     at Object.<anonymous> (/home/edison/.nvm/versions/node/v12.5.0/lib/node_mod
0|server   |     at Module._compile (internal/modules/cjs/loader.js:776:30) {
0|server   |   code: 'MODULE_NOT_FOUND',
```

- Open your Participant Guide.
- Review the information provided.
- Install, configure and monitor PM2

Hands On: Installing, Configurating and Monitoring PM2

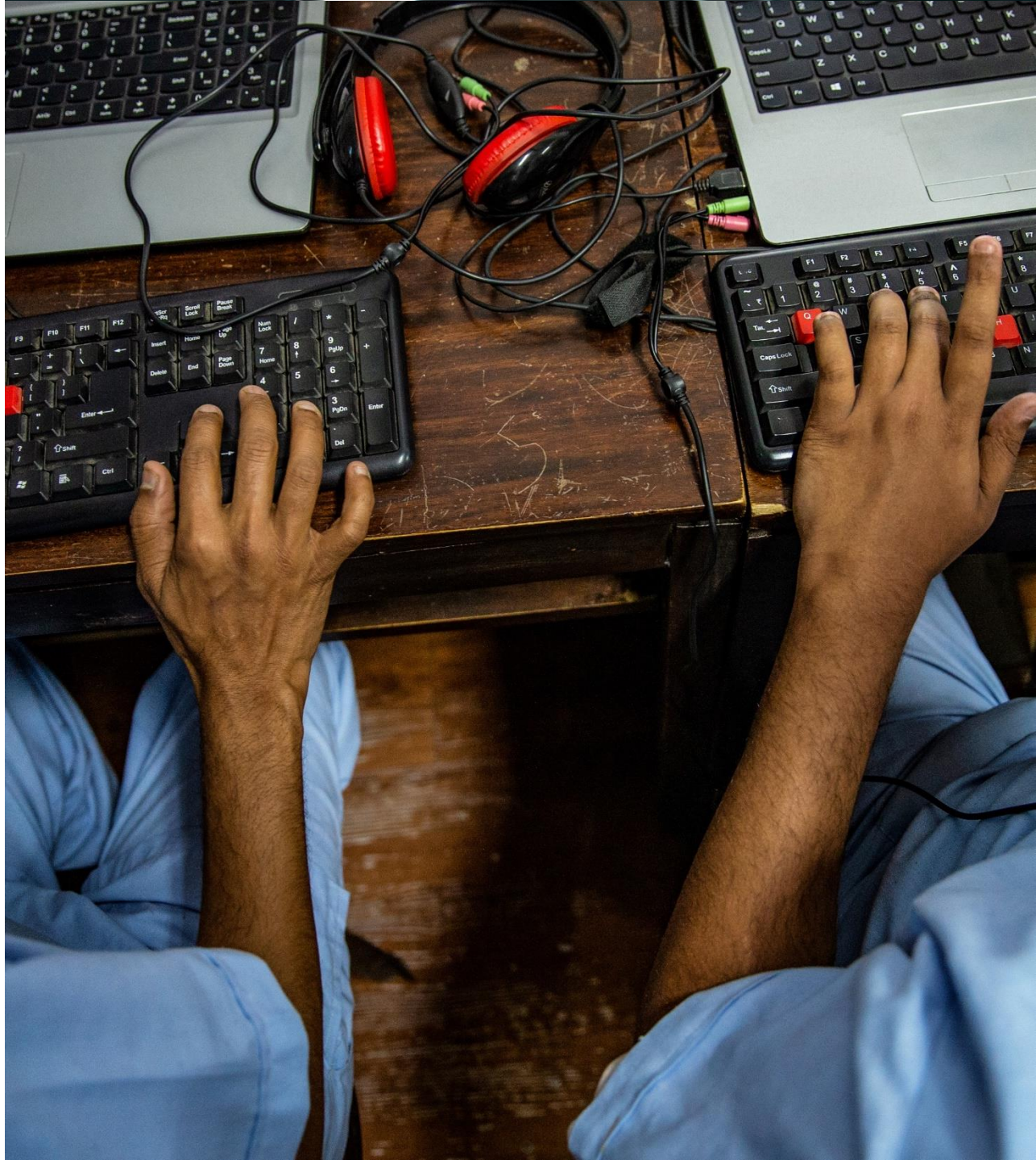
- Open your Participant Guide.
- Review the information provided.

Hands On: Using PM2 to Serve Angular Web Apps

Lesson Summary

In this lesson, we learned to:

- Install and configure PM2
- Monitor and review logs with apps running under PM2
- Configure PM2 Azure Web Apps Linux and serve static files



Microsoft

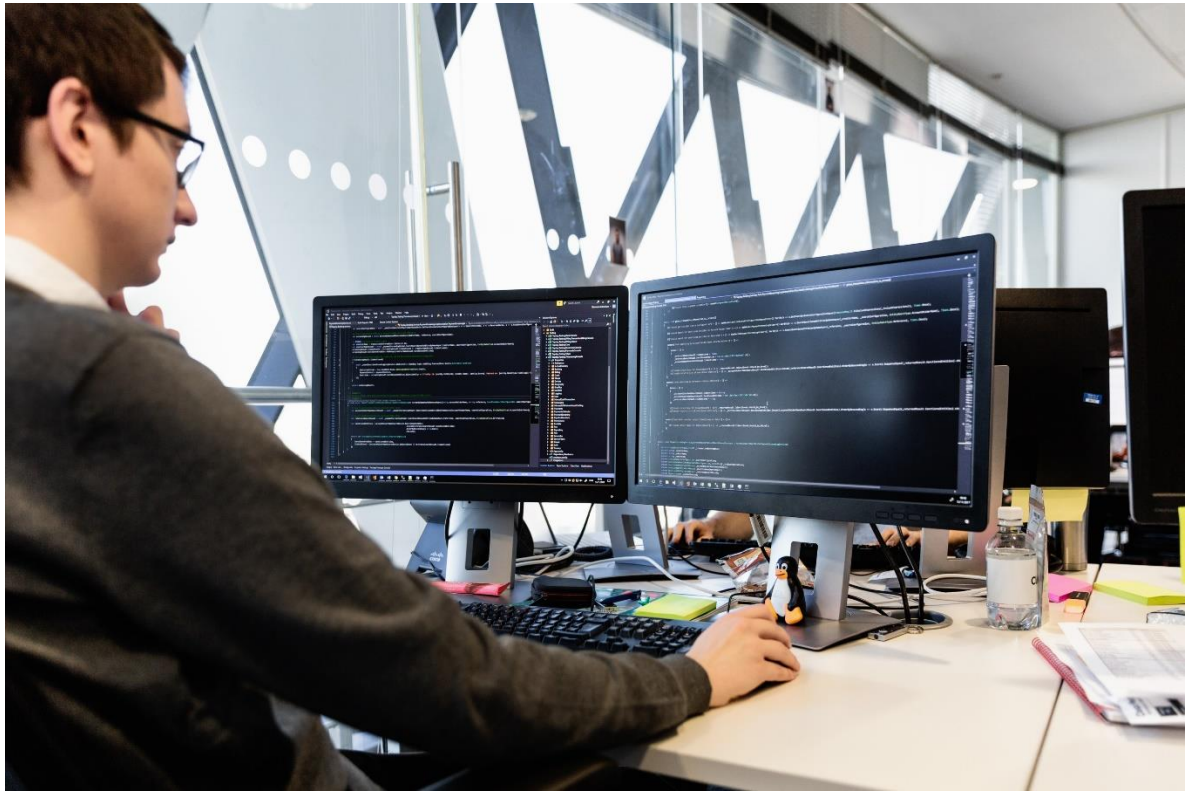
How to Install and Configure Nginx

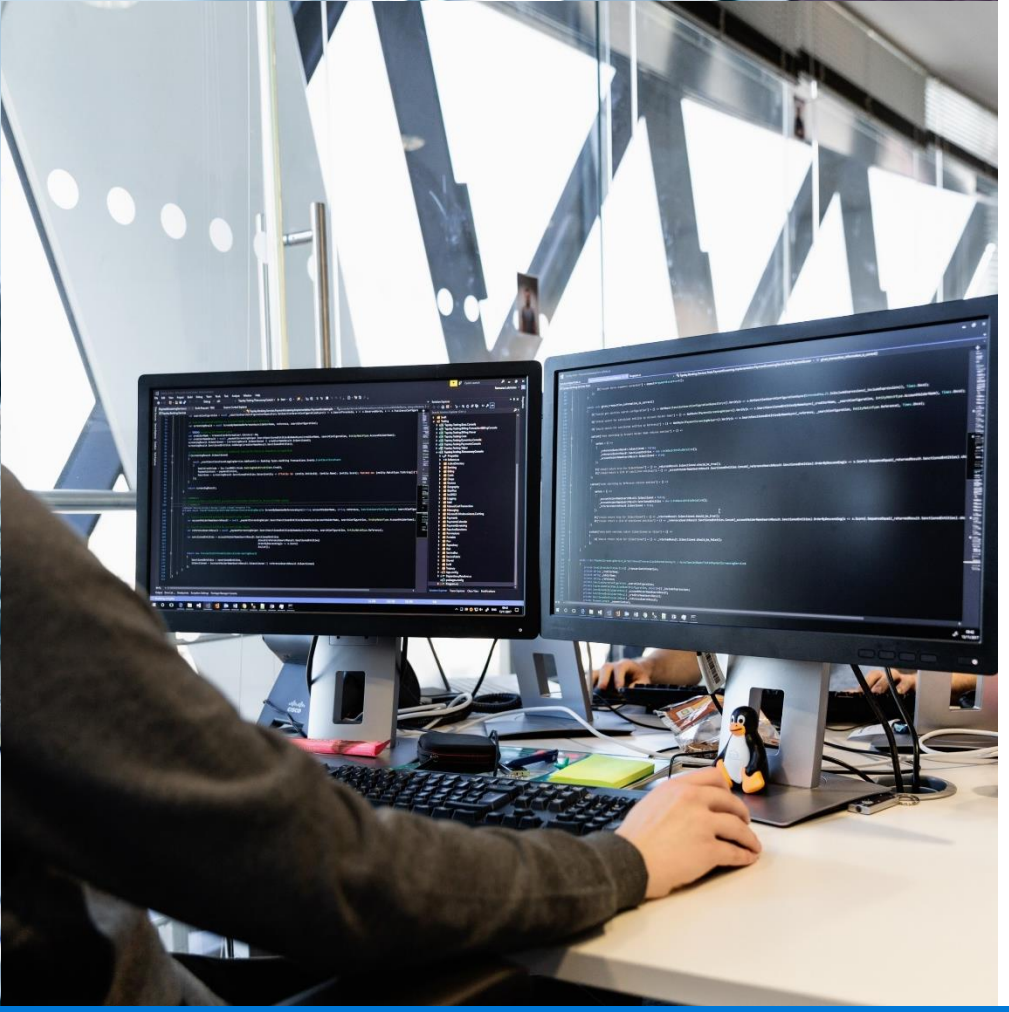
Module Objectives

- Install Nginx as a web server & understand the default file structure.
- Configure Nginx with Virtual Hosts
- Configured Nginx as a Reverse Proxy with Node Express

Overview

Nginx is a widely used web server known for its versatility & efficiency. While it operates as an HTTP Web Server it can also function as a reverse proxy, mail proxy server & TCP/UDP proxy server. Similar web servers like Apache may be resource-heavy creating new processes/threads to handle each new connection. Instead, Nginx has a master process operating single-threaded & asynchronously; this process controls multiple workers to handle concurrent connections which are known to scale well under high load in a non-blocking manner.





- Open your Participant Guide.
- Review the information provided.
- Install and run Nginx

Hands On: Installing & Running Nginx

Lesson Summary

In this lesson, we learned to:

- Install Nginx as a web server
- Understand the default file structure.
- Configure Nginx with Virtual Hosts
- Configured Nginx as a Reverse Proxy with Node Express

Microsoft

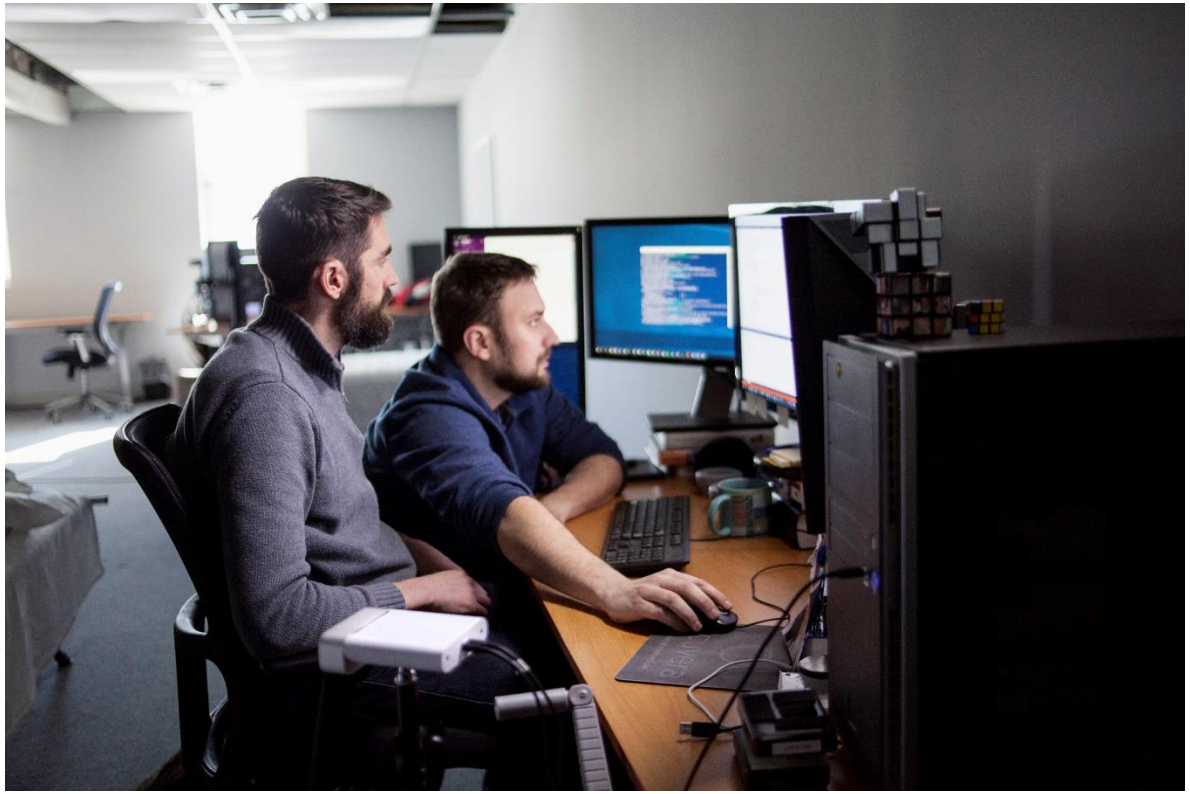
Configuring Web.config for NodeJS Apps on Windows

Module Objectives

- Configure web.config Python Apps on App Service Windows.
- Understand Python Site Extensions on App Service Windows.

Overview

This lesson covers how to configuration Python applications on Azure App Service on Windows as well as other common configurations for different Python frameworks.



Sample Web.config

```
<?xml version="1.0" encoding="utf-8"?>
<!--
     This configuration file is required if iisnode is used to run node processes behind
     IIS or IIS Express.  For more information, visit:

     https://github.com/tjanczuk/iisnode/blob/master/src/samples/configuration/web.config
-->

<configuration>
  <system.webServer>
    <!-- Visit http://blogs.msdn.com/b/windowsazure/archive/2013/11/14/introduction-to-websock
    <webSocket enabled="false" />
    <handlers>
      <!-- Indicates that the server.js file is a node.js site to be handled by the iisnode mo
      <add name="iisnode" path="server.js" verb="*" modules="iisnode"/>
    </handlers>
    <rewrite>
      <rules>
        <!-- Do not interfere with requests for node-inspector debugging -->
        <rule name="NodeInspector" patternSyntax="ECMAScript" stopProcessing="true">
          <match url="^server.js\/debug[\/]?" />
        </rule>

        <!-- First we consider whether the incoming URL matches a physical file in the /public
        <rule name="StaticContent">
          <action type="Rewrite" url="public{REQUEST_URI}"/>
        </rule>

        <!-- All other URLs are mapped to the node.js site entry point -->
```

Lesson Summary

In this lesson, we learned to:

Setup web.config for NodeJS apps on Windows.

Identify some web.config samples

Microsoft

Troubleshooting Performance Issues for NodeJS Using V8 Profiler on Windows

Module Objectives

- Use XDebug and review output to determine bottlenecks for PHP on Windows.

Overview

This lesson introduces you to troubleshooting performance issues for NodeJS on Windows.

Finding Memory Leaks and CPU Usage in Azure Node.js Web App

Slow application performance issues tend to be challenging to troubleshoot regardless of the platform in which your application is running.

This is due in great part to the sometimes random nature of these issues.

These types of issues also often do not result in a specific error being logged.

If you think your node.js application is running slow and takes more than few seconds to receive response. Below info may help you analyze where it's taking longer time and also checks for memory leaks.

There are many node.js modules to accomplish this(notably)

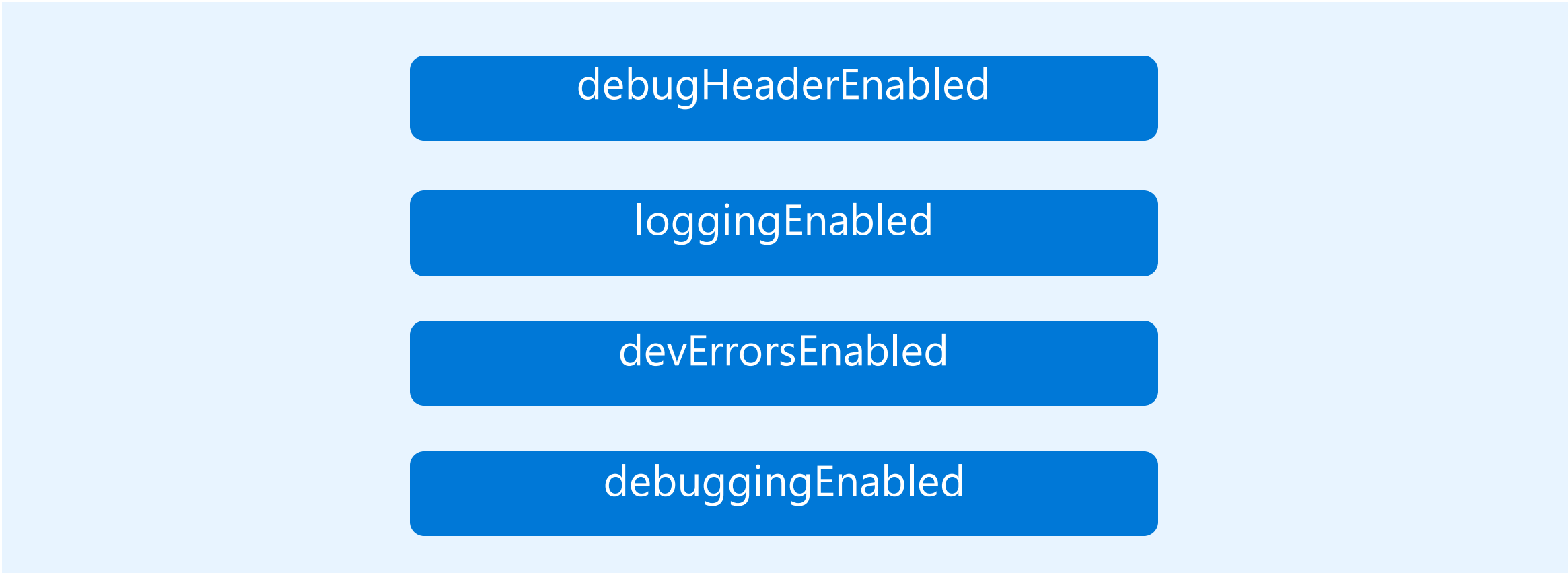
- v8-profiler
- nodetime(app dynamics)
- look

Best Practices and Troubleshooting Guide for Node Applications on Azure App Service Windows

iisnode schema settings, here are the most common settings and their use:

- nodeProcessCountPerApplication
- nodeProcessCommandLine
- maxConcurrentRequestsPerProcess
- maxNamedPipeConnectionRetry
- namedPipeConnectionRetryDelay
- logDirectory
- flushResponse
- watchedFiles
- recycleSignalEnabled
- idlePageOutTimePeriod

Best practices and troubleshooting guide for node applications on Azure App Service Windows Part 2



Scenarios Recommendations/Troubleshooting

My node application is consuming too much CPU

My node application is making excessive outbound calls

My node application is consuming too much memory

My node.exe's are getting killed randomly

My node application does not start

My node application crashed

My node application takes too much time to start (Cold Start)

IISNODE Http Status and Substatus

Http Status	Http Substatus	Possible Reason
500	1000	There was some issue dispatching the request to IISNODE – Check if node.exe was started. Node.exe could have crashed when starting. Check your web.config configuration for errors.
500	1001	- Win32Error 0x2 - App is not responding to the URL. Check the URL rewrite rules or check if your express app has the correct routes defined. - Win32Error 0x6d – named pipe is busy – Node.exe is not accepting requests because the pipe is busy. Check high cpu usage. - Other errors – check if node.exe crashed.
500	1002	Node.exe crashed – check d:\home\LogFiles\logging-errors.txt for stack trace.
500	1003	Pipe configuration Issue – The named pipe configuration is incorrect.
500	1004-1018	There was some error while sending the request or processing the response to/from node.exe. Check if node.exe crashed. check d:\home\LogFiles\logging-errors.txt for stack trace.

IISNODE Http Status and Substatus 2

Http Status	Http Substatus	Possible Reason
503	1000	Not enough memory to allocate more named pipe connections. Check why your app is consuming so much memory. Check maxConcurrentRequestsPerProcess setting value. If it's not infinite and you have many requests, increase this value to prevent this error.
503	1001	Request could not be dispatched to node.exe because the application is recycling. After the application has recycled, requests should be served normally.
503	1002	Check win32 error code for actual reason – Request could not be dispatched to a node.exe.
503	1003	Named pipe is too Busy – Verify if node.exe is consuming excessive CPU
503	1000	Not enough memory to allocate more named pipe connections. Check why your app is consuming so much memory. Check maxConcurrentRequestsPerProcess setting value. If it's not infinite and you have many requests, increase this value to prevent this error.

Lesson Summary

In this lesson, we learned to:

- Find Memory Leaks and CPU Usage in Azure Node.js Web App
- Identify IISNODE Http Status and Substatus
- Troubleshoot most common scenarios