

CONINT Semester Project

S4-CONINT

**Amelia Maier
Nick Müllner
Sophie Peirl
Rene Rumetshofer**

Table of Contents

Table of Contents	2
CONINT Semester Project	3
1. Applications + Docker Files	4
1.1 Backend	4
1.2 Frontend	9
2. Version Control	15
3. Infrastructure (AWS)	16
3.1 Network setup	16
3.2 EC2 setup	19
3.3 Docker host configuration	22
3.4 NGINX Proxy manager	24
3.5 RDS setup	26
4. Docker Hub	29
5. CI Servers	30
5.1 GitHub Actions	30
5.2 Jenkins	32
6. Code Quality Server	37
6.1 SonarQube container setup	37
6.2 SAST configuration	39
7. Security Scan Server	45
8. Feature Toggle & A/B Test Server	48
9. Each Stage of the Pipeline	50
9.1 GitHub actions	50
9.2 Jenkins	57
10. Features and Refined User Stories	64
11. A/B Testing	67
12. Blue/Green Deployment	70
13. Logging and Monitoring (Outlook)	73

CONINT Semester Project

Participants:

Nick Müllner – if23b503@technikum-wien.at

Sophie Peirl – if23b020@technikum-wien.at

Rene Rumetshofer – if24b006@technikum-wien.at

Amelia Maier - if23b184@technikum-wien.at

Brief description of used technology stack:

Backend: Node.js application with Fastify

Frontend: Static webserver with nginx

Source control: GitHub

Cloud-hosted CI: GitHub Actions

Self-hosted CI: Jenkins on AWS

Task	Assigned person
Frontend – Implementation & Dockerization & tests	Nick Müllner
Backend – Implementation & Dockerization & tests	Rene Rumetshofer
E2E & performance testing setup	Sophie Peirl
GitHub Actions pipeline setup	Rene Rumetshofer
AWS infrastructure setup	Rene Rumetshofer
PostHog setup & Frontend – A/B integration	Amelia Maier
Sonarqube setup	Rene Rumetshofer
Jenkins setup	Nick Müllner
Jenkins pipeline setup	Nick Müllner
Blue/Green Deployment setup	Sophie Peirl
Snyk setup	Rene Rumetshofer
Documentation polishing	Amelia Maier

1. Applications + Docker Files

- Explanation of the chosen frameworks (Vue, React, or Express)
- Dockerfile instructions, ports, environment variables

1.1 Backend

Used packages and technologies:

- Docker compose for development
- Prettier for formatting code
- Postgrator for DB migrations

Database credentials are stored in the local docker.env file at the project root level. The variables are loaded by the docker-compose file. Because the setup on development and on production is the same, there is no need to split up the environment:

```
"scripts": {
  "eslint": "eslint src/",
  "sonar-scanner": "sonar-scanner -Dsonar.host.url=$SONAR_HOST_URL -Dsonar.login=$SONAR_TOKEN",
  "snyk-auth": "snyk auth --auth-type=token $SNYK_TOKEN",
  "snyk": "snyk test",
  "jest": "jest --coverage",
  "start-compose-v1": "docker-compose -f ./docker-compose.yaml up -d --build backend",
  "start-compose-v2": "docker compose -f ./docker-compose.yaml up -d --build backend"
},
```

Testing is set up by first installing the following packages as dev dependencies:

- "@eslint/js": "^9.31.0",
- "@babel/core": "^7.28.0",
- "@babel/preset-env": "^7.28.0",
- "eslint": "^9.31.0",
- "babel-jest": "^30.0.4",
- "globals": "^16.3.0",
- "jest": "^30.0.4",
- "jest-environment-node": "^30.0.4",
- "prettier": "^3.6.2",
- "snyk": "^1.1297.3",
- "supertest": "^7.1.3",
- "sonar-scanner": "^3.1.0"

Setup of jest.config.js:

```
export default {  no usages  & Rene Rumetshofer +1
  testEnvironment: 'node',
  testMatch: ['<rootDir>/test/jest/*.spec.js'],
  transform: {
    '^.+\\.(js|jsx)$': 'babel-jest',
  },
};
```

Setup of babel.config.js:

```
backend > B babel.config.js > ...
1  export default {
2    |   presets: ['@babel/preset-env'],
3    |   plugins: ['@babel/plugin-transform-modules-commonjs'],
4  };
```

Setup of eslint.config.mjs:

```
import js from '@eslint/js';
import globals from 'globals';
import { defineConfig } from 'eslint/config';

export default defineConfig( args: [ no usages  ↴
{
  files: ['**/*.{js,mjs,cjs}'],
  plugins: { js },
  extends: ['js/recommended'],
  languageOptions: {
    globals: {
      ...globals.browser,
      process: 'readonly',
      Buffer: 'readonly',
    },
  },
},
{
  files: ['**/*spec.js'],
  languageOptions: {
    globals: globals.jest,
  },
},
);
];
```

Integration testing is being setup by installing the dev dependency “supertest” and executed through jest. Unit tests (spec files) and integration tests are specified under the folder “tests/jest”.

The application first runs migrations via the migration service and only afterwards starts Fastify by listening on all interfaces. Before starting, all routes and the database connector are registered.

```
import Fastify from 'fastify';
import cors from '@fastify/cors';
import dbConnector from './db/postgres-connector.js';
import notesRoutes from './routes/notes.routes.js';
import migrate from './db/migration.service.js';

try {
  await migrate();
} catch (err) {
  console.error('Migrations failed with error; ', err);
  process.exit(1);
}

const fastify = Fastify({
  logger: true,
});

await fastify.register(cors, {
  origin: '*',
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
});

fastify.register(dbConnector);
fastify.register(notesRoutes);

fastify.listen({ port: 3000, host: '0.0.0.0' }, function (err, address) {
  if (err) {
    fastify.log.error(err);
    process.exit(1);
  }
});
```

Routes are stored under the directory “routes”, database services including migration files under “db” and general services under “services”. Please see the repository for details.

The migration service establishes a DB connection by using the environment variables and then runs all SQL files under the sub-directory “migrations”:

```
import pg from 'pg';
import Postgrator from 'postgrator';
import path from 'node:path';
import { fileURLToPath } from 'url';
import { dirname } from 'path';

async function migrate() {
  const client = new pg.Client({
    host: process.env.DB_HOST,
    port: process.env.DB_PORT,
    database: process.env.DB_NAME,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
  });

  await client.connect();

  const __filename = fileURLToPath(import.meta.url);
  const __dirname = dirname(__filename);
  const postgrator = new Postgrator({
    migrationPattern: path.join(__dirname, '/migrations/*'),
    driver: 'pg',
    database: process.env.DB_NAME,
    schemaTable: 'migrations',
    currentSchema: 'public', // Postgres and MS SQL Server only
    execQuery: (query) => client.query(query),
  });

  const result = await postgrator.migrate();

  if (result.length === 0) {
    console.log(
      'No migrations run for schema "public". Already at the latest one.'
    );
  }

  console.log('Migration done.');
  await client.end();
}

export default migrate;
```

There is only one route file called "notes.routes.js". The functionality of such files can be explained via the first routes:

```
import {
  encryptNote,
  generateUUID,
  decryptNote,
} from '../services/crypto.service.js';

async function routes(fastify, options) {
  fastify.get('/notes', async (request, reply) => {
    fastify.pg.query(
      'SELECT notes_uuid, title FROM notes',
      function onResult(err, result) {
        reply.send(err || result?.rows);
      }
    );
    return reply;
  });

  const getNoteOpts = {
    schema: {
      params: {
        type: 'object',
        required: ['uuid'],
        properties: {
          key: { type: 'string' },
        },
      },
      query: {
        type: 'object',
        required: ['key'],
        properties: {
          key: { type: 'string' },
        },
      },
    },
  };
  fastify.get('/notes/:uuid', getNoteOpts, async (request, reply) => {
    console.log(request.query);
    fastify.pg.query(
      'SELECT content FROM notes WHERE notes_uuid = $1',
      [request.params.uuid],
      function onResult(err, result) {
        if (err) {
          reply.code(500).send('Error while fetching note');
          return;
        }
        if (result.rows.length === 0) {
          reply.code(404).send('Note not found');
          return;
        }
        try {
          reply.send(decryptNote(request.query.key, result.rows[0].content));
        } catch (err) {
          reply.code(403).send('Key is invalid');
        }
      }
    );
    return reply;
  });
}
```

The first route simply fetches all notes and returns the UUIDs and titles only. The second GET route takes a path parameter and a query variable to select a specific note by UUID and provide a key (request.query.key).

The Dockerfile uses node version 24, installs all packages, copies all files from src and starts the backend. The environment file is not specified here as the env variables will later be passed by using Docker compose environment variables.

```
FROM node:24-alpine

WORKDIR /app

COPY package*.json .
RUN npm ci --omit=dev --ignore-scripts
COPY src .

EXPOSE 3000

CMD ["node", "index.js"]
```

1.2 Frontend

The Frontend is pretty similar to the Backend: also using jest for unit testing, docker compose for deployment and a Dockerfile for a custom Docker Image. There is no env file necessary, since the Frontend is a basic nginx container which serves a static website. Compared to the Backend, the Frontend also contains e2e testing through playwright and performance testing through k6:

```
"scripts": {
  "eslint": "eslint src/",
  "sonar-scanner": "sonar-scanner -Dsonar.host.url=$SONAR_HOST_URL -Dsonar.login=$SONAR_TOKEN",
  "snyk-auth": "snyk auth --auth-type=token $SNYK_TOKEN",
  "snyk": "snyk test",
  "jest": "jest --coverage",
  "playwright": "playwright test",
  "k6": "k6 run test/k6/load-test.js",
  "start-compose-v1": "docker-compose -f ./docker-compose.yaml up -d --build frontend",
  "start-compose-v2": "docker compose -f ./docker-compose.yaml up -d --build frontend"
},
```

The following development dependencies are necessary for testing:

- "@eslint/js": "^9.31.0",
- "@jest/globals": "^30.0.4",
- "@playwright/test": "^1.54.1",
- "eslint": "^9.31.0",
- "globals": "^16.3.0",
- "jest": "^30.0.4",
- "jest-environment-jsdom": "^30.0.4",
- "k6": "^0.0.0",
- "playwright": "^1.54.1",
- "snyk": "^1.1297.3",
- "sonar-scanner": "^3.1.0"

Setup of jest.config.js:

```
module.exports = {
  testEnvironment: "jsdom",
  testMatch: ["<rootDir>/test/jest/*.spec.js"],
};
```

Setup of playwright.config.js:

```
const { defineConfig } = require("@playwright/test");

module.exports = defineConfig({
  use: {
    browserName: "chromium",
    headless: true,
    launchOptions: {
      args: ["--no-sandbox", "--disable-setuid-sandbox"],
    },
    testDir: "./test/playwright",
  });
});
```

Setup of eslint.config.mjs:

```
import js from "@eslint/js";
import globals from "globals";
import { defineConfig } from "eslint/config";

export default defineConfig(args: [ no usages &
{
  files: ["**/*.{js,mjs,cjs}"],
  plugins: { js },
  extends: ["js/recommended"],
  languageOptions: {
    globals: {
      ...globals.browser,
    },
  },
  {
    files: ["**/tests/jest/*.js"],
    languageOptions: {
      globals: globals.jest,
    },
  },
],
]);
```

Tests are found under the following directory:

- test/jest → Unit testing
- test/k6 → performance testing
- test/playwright → e2e testing

A simple structured index.html provides a clean interface for creating, viewing and deleting notes. Furthermore, index.js and posthog.js are linked within the file. To ensure Eventlisteners are created after the DOM has loaded, a small check is also included:

```
<h2>Alle Notizen</h2>
<div id="notes"></div>

<h2>Neue Notiz erstellen</h2>
<form id="newNoteForm">
  <label for="newTitle">Titel</label>
  <input id="newTitle" placeholder="Titel" required type="text" />
  <label for="newContent">Inhalt</label>
  <textarea id="newContent" placeholder="Inhalt" required></textarea>
  <label for="newKey">Verschlüsselungs-Key</label>
  <input id="newKey" placeholder="Verschlüsselungs-Key" required type="text" />
  <button id="createNote" type="submit">Erstellen</button>
</form>

<script src="posthog.js"></script>
<script src="index.js"></script>
<script>
  if (document.readyState === "loading") {
    document.addEventListener("DOMContentLoaded", () => {
      initializeDOMInteractions(true);
    });
  } else {
    initializeDOMInteractions(true);
  }
</script>
```

To enable several environments (local, staging, production, Jenkins), a workaround was used which relies on the URL, which is set on the used browser, dynamically determining the API address:

```
const API_HOSTS :{} = {
  localhost: "http://localhost:3000/api",
  "staging.conint-securenotes.online":
    "https://staging.conint-securenotes.online/api",
  "prod.conint-securenotes.online":
    "https://prod.conint-securenotes.online/api",
  "frontend-green": "http://backend-green:3000/api",
};

const hostname :string = window.location.hostname;
const API = API_HOSTS[hostname];
```

Viewing and deleting notes are defined in own methods, creating notes is defined as an Eventlistener which is added after DOM loaded:

```
async function loadNotes() :Promise<void> { ... }

async function loadNote(uuid) :Promise<void> { ... }

async function deleteNote(uuid) :Promise<void> { ... }

async function initializeDOMInteractions(isLoadNotes) :Promise<void> { Show usages & Ni
  document
    .getElementById( elementId: "newNoteForm")
    .addEventListener( type: "submit", listener: async (e :SubmitEvent) :Promise<void> => {
      e.preventDefault();
      const title = document.getElementById( elementId: "newTitle").value;
      const content = document.getElementById( elementId: "newContent").value;
      const key = document.getElementById( elementId: "newKey").value;

      const res :Response = await fetch( input: `${API}/notes`, init: {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify( value: { title, content, key })
      });

      if (res.ok) {
        alert("Notiz erstellt!");
        document.getElementById( elementId: "newNoteForm").reset();
        await loadNotes();
      } else {
        alert("Fehler beim Erstellen der Notiz.");
      }
    });
}
```

Testing with jest, k6 and playwright was a challenge, since all of them need URLs to Frontend and Backend, but they differ in environment passed variable handling. Jest exports the methods from the index.js and mocks the fetches to the API:

```
const {
  initializeDOMInteractions,
  loadNotes,
  loadNote,
  deleteNote,
} = require("../src/index.js");

beforeEach(() :void => { ➜ Nick Müllner
  document.body.innerHTML =
    <form id="newNoteForm">
      <input id="newTitle" />
      <input id="newContent" />
      <input id="newKey" />
      <button type="submit">Submit</button>
    </form>
    <div id="notes">
      <div class="note">
        <strong>Test Note 3</strong><br>
        UUID: 789<br>
        <input type="text" placeholder="Key eingeben" id="key-789">
        <button id="show-789">Anzeigen</button>
        <button id="del-789">Löschen</button>
        <pre id="content-789"></pre>
      </div>
    </div>
  ;
}

global.fetch = jest.fn(() :any | Promise<...> =>
  Promise.resolve( value: {
    json: () :any | Promise<...> => Promise.resolve( value: [] ),
  }),
);

initializeDOMInteractions( isLoadNotes: false);
});

test("loadNotes should load notes and display them", async () :Promise<void> => {
  fetch.mockResolvedValueOnce({
    ok: true,
    json: async () :Promise<[{notes_uuid: string, title: string}]> => [
      { notes_uuid: "123", title: "Test Note 1" },
      { notes_uuid: "456", title: "Test Note 2" },
    ],
  });

  await loadNotes();

  expect(document.querySelectorAll( selectors: ".note").length).toBe( expected: 2);
  expect(document.querySelector( selectors: ".note").textContent).toContain(
    expected: "Test Note 1",
  );
});
```

K6 is using its own environment, making it necessary to pass variables in a specific way:

```
const BASE_URL = process.env.BACKEND_GREEN || "http://localhost:3000/api";

export default function (): void {
  no usages ↗ Sophie Peirl +2
  group("Notiz erstellen → abrufen → löschen", () : void => {
    // 1. Notiz erstellen
    const payload : string = JSON.stringify( value: {
      title: `Titel ${__VU}-${__ITER}`,
      content: "Dies ist ein Loadtest-Inhalt",
      key: "kótestkey",
    });
    const headers : {Content-Type: string} = { "Content-Type": "application/json" };
    const postRes = http.post(`${BASE_URL}/notes`, payload, { headers });
    check(postRes, {
      "Notiz erfolgreich erstellt": (res) : boolean => res.status === 200,
    });
    let uuid = postRes.json().notes_uuid;
  });
}
```

Finally, we have playwright:

```
const BASE_URL : string = process.env.FRONTEND_GREEN || "http://localhost:80";

test.beforeEach( inner: async ({ page }: Page) : Promise<void> => {
  await page.goto(BASE_URL);
});

test(title: "1. Seite lädt und zeigt Notizliste an (auch wenn leer)", body: async ({ page }: Page) : Promise<void> => {
  const notesContainer : Locator = page.locator({ selector: "#notes" });
  const count : number = await notesContainer.locator({ selectorOrLocator: ".note" }).count();
  expect(count).toBeGreaterThanOrEqual(expected: 0);
});
```

Summarized, we use either dynamically or “passed via environment variables” URLs to Frontend and Backend to ensure correct paths on different environments. Like this, local development and testing is still possible while CICD tools can manipulate URLs for their needs.

A simple Dockerfile is used for creating the image. Using nginx 1.29, copying the src folder and exposing on port 80:

```
FROM nginx:1.29-alpine

COPY src /usr/share/nginx/html

EXPOSE 80
```

2. Version Control

- Branching strategy (main, feature branches, deploy/production branch)

For version control, GitHub is used. <https://github.com/ReneRumetschofer/CONINT>

Trunk-based development is used, so feature branches should be short lived and often rebased on main. Merges back into main are only allowed via pull requests. The main branch is protected from direct pushes.

As it's a requirement for this project, deployments to the staging and prod environments, as well as steps leading up to this, will only take place when a commit is pushed onto the branch called "production". This branch should not have direct commits and can only be updated via pull requests. This is also enforced via a branch protection rule.

3. Infrastructure (AWS)

- EC2 instance configuration
- RDS configuration for PostgreSQL
- Network details and security groups

3.1 Network setup

Setup VPC as 10.0.0.0/24 virtual network:

VPC erstellen Info

Eine VPC ist ein isolierter Teil der AWS-Cloud, der von AWS-Objekten wie Amazon-EC2-Instances gefüllt wird.

VPC-Einstellungen

Zu erstellende Ressourcen Info

Erstellen Sie nur die VPC-Ressource oder die VPC und andere Netzwerkressourcen.

Nur VPC

VPC und mehr

Namens-Tag – optional

Erstellt ein Tag mit dem Schlüssel 'Name' und einem von Ihnen angegebenen Wert.

conint-vpc

IPv4-CIDR-Block Info

- Manuelle IPv4 CIDR-Eingabe
- IPAM-zugewiesener IPv4-CIDR-Block

IPv4-CIDR

10.0.0.0/24

Die CIDR-Blockgröße muss zwischen /16 und /28 liegen.

IPv6-CIDR-Block Info

- Kein IPv6 CIDR-Block
- IPAM-zugewiesener IPv6-CIDR-Block
- Von Amazon bereitgestellter IPv6 CIDR-Block
- IPv6 CIDR in meinem Besitz

Tenancy Info

Standard



A public subnet is added which gets half of all IPs, therefore using network mask 10.0.0.0/25:

Subnetzeinstellungen

Geben Sie die CIDR-Blöcke und Availability Zone für das Subnetz an.

Subnetz 1 von 1

Subnetz-Name

Ein Tag mit dem Schlüssel 'Name' und einem von Ihnen angegebenen Wert erstellen.

conint-public

Der Name kann bis zu 256 Zeichen lang sein.

Verfügbare Zone Info

Wählen Sie die Zone aus, in der sich Ihr Subnetz befindet, oder lassen Sie Amazon eine für Sie auswählen.

Keine Präferenz

IPv4-VPC-CIDR-Block Info

Wählen Sie den IPv4-CIDR-Block der VPC für das Subnetz. Der IPv4-CIDR des Subnetzes muss innerhalb dieses Blocks liegen.

10.0.0.0/24

IPv4-Subnetz-CIDR-Block

10.0.0.0/25

128 IPs

< > ^ v

▼ Tags – optional

Schlüssel

Q Name

Wert - optional

Q conint-public

X

Entfernen

Neues Tag hinzufügen

Sie können 49 weitere Tags hinzufügen

Entfernen

Creating a internet gateway for internet connectivity:

Internet-Gateway erstellen Info

Ein Internet-Gateway ist ein virtueller Router, der eine VPC mit dem Internet verbindet. Geben Sie zum Erstellen eines neuen Internet-Gateways den Namen für das unten stehende Gateway an.

Internet-Gateway-Einstellungen

Namens-Tag

Erstellt ein Tag mit dem Schlüssel 'Name' und einem von Ihnen angegebenen Wert.

conint-public-igw

Tags – optional

Ein Tag ist eine Bezeichnung, die Sie einer AWS-Ressource zuweisen. Jedes Tag besteht aus einem Schlüssel und einem optionalen Wert. Mithilfe von Tags können Sie Ihre Ressourcen durchsuchen und filtern oder Ihre AWS-Kosten nachverfolgen.

Schlüssel

Q Name

Wert - optional

Q conint-public-igw

X

Entfernen

Neues Tag hinzufügen

Sie können 49 weitere Tags hinzufügen

Abbrechen

Internet-Gateway erstellen

This IGW is now being attached to the VPC:

An VPC anfügen (igw-0ef09ec1dbbc7d029) Info

VPC

Schließen Sie ein Internet-Gateway an eine VPC an, um die Kommunikation mit dem Internet zu ermöglichen. Geben Sie unten die VPC an, die Sie anfügen möchten.

Verfügbare VPCs

Fügen Sie das Internet-Gateway dieser VPC an.

vpc-0bd7cd0e3b0a09cbe X

► AWS-Befehlszeilenschnittstellenbefehl

Abbrechen Internet-Gateway anfügen

The last piece for getting connectivity is setting up a route that points to the internet gateway:

Routen bearbeiten

Bestimmungsort	Ziel	Status	Propagiert
10.0.0.0/24	local	Aktiv	Nein
<input type="text"/> 0.0.0.0/0 X	Internet-Gateway	Aktiv	Entfernen
	<input type="text"/> igw-0ef09ec1dbbc7d029 X		

Route hinzufügen Abbrechen Vorschau Änderungen speichern

3.2 EC2 setup

A new EC2 instance is created that will be used for hosting docker all needed docker containers for this project.

Eine Instance starten Info

Mit Amazon EC2 können Sie virtuelle Maschinen oder Instances erstellen, die in der AWS Cloud ausgeführt werden. Führen Sie für einen schnellen Einstieg die folgenden einfachen Schritte aus.

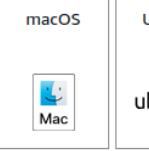
Name und Tags Info

Name Weitere Tags hinzufügen

▼ Anwendungs- und Betriebssystemabbilder (Amazon Machine Image) Info

Ein AMI ist eine Vorlage, die die Softwarekonfiguration (Betriebssystem, Anwendungsserver und Anwendungen) enthält, die zum Starten Ihrer Instance benötigt wird. Suchen Sie nach AMIs, wenn Sie unten nicht sehen, wonach Sie suchen

AMI aus Katalog **Schnellstart**

Weitere AMIs durchsuchen 
Einschließlich AMIs von AWS, Marketplace und der Community

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI ami-02457590d33d576c3 (64-Bit (x86), uefi-preferred) / ami-095f54910906baa61 (64-Bit (Arm), uefi) Virtualisierung: hvm ENA-aktiviert: true Stammgerätytyp: ebs	Zur kostenlosen Nutzung berechtigt 
--	--

Beschreibung
Amazon Linux 2023 ist ein modernes Linux-basiertes Allzweck-Betriebssystem mit 5 Jahren Langzeit-Support. Es ist für AWS optimiert und bietet eine sichere, stabile und leistungsstarke Ausführungsumgebung für die Entwicklung und Ausführung Ihrer Cloud-Anwendungen.

Amazon Linux 2023 AMI 2023.7.20250527.1 x86_64 HVM kernel-6.1

▼ Instance-Typ Info | Lassen Sie sich beraten

Instance-Typ

t2.medium
Familie: t2 2 vCPU 4 GiB Arbeitsspeicher Aktuelle Generation: true
On-Demand Ubuntu Pro Basis Preise: 0.0499 USD pro Stunde On-Demand Linux Basis Preise: 0.0464 USD pro Stunde
On-Demand RHEL Basis Preise: 0.0752 USD pro Stunde On-Demand Windows Basis Preise: 0.0644 USD pro Stunde
On-Demand SUSE Basis Preise: 0.1464 USD pro Stunde

Alle Generationen 
Instance-Typen vergleichen

Für AMIs mit vorinstallierter Software fallen zusätzliche Kosten an

“t2.medium” is used as instance type, as this brings 2 vCPU cores with 4 GiB of RAM which should suffice for running a bunch of docker containers fluidly.

A new keypair is created for SSH access:

Schlüsselpaar erstellen



Schlüsselpaarname

Mit Schlüsselpaaren können Sie eine sichere Verbindung mit Ihrer Instance herstellen.

Der Name darf bis zu 255 ASCII-Zeichen enthalten. Er darf keine Leerzeichen am Anfang oder am Ende enthalten.

Typ des Schlüsselpaars

RSA

Mit RSA verschlüsseltes Paar aus
privatem und öffentlichem Schlüssel

ED25519

Mit ED25519 verschlüsseltes privates
und öffentliches Schlüsselpaar

Dateiformat für privaten Schlüssel

.pem

Zur Verwendung mit OpenSSH

.ppk

Zur Verwendung mit PuTTY



Wenn Sie dazu aufgefordert werden, speichern Sie den privaten Schlüssel
an einem sicheren und zugänglichen Ort auf Ihrem Computer. **Sie
benötigen es später, um eine Verbindung mit Ihrer Instance herzustellen.**

[Weitere Informationen](#)

[Abbrechen](#)

[Schlüsselpaar erstellen](#)

Networking is configured to use the previously created VPC and public subnet. A new security group is being created that for configuring the instance contains one incoming SSH rule for the IP address of the configuring user:

▼ Netzwerkeinstellungen [Info](#)

VPC – Pflichtfeld | [Info](#)

vpc-0bd7cd0e3b0a09cbe (conint-vpc)
10.0.0.0/24

Subnetz | [Info](#)

subnet-0066f80617050e383
VPC: vpc-0bd7cd0e3b0a09cbe Besitzer: 665624292588 Availability Zone: us-east-1a
Zonentyp: Availability Zone IP-Adressen verfügbar: 123 CIDR: 10.0.0.0/25

conint-public

Neues Subnetz erstellen

Öffentliche IP automatisch zuweisen | [Info](#)

Deaktivieren

Firewall (Sicherheitsgruppen) | [Info](#)

Eine Sicherheitsgruppe ist eine Reihe von Firewall-Regeln, die den Datenverkehr für Ihre Instance steuern. Fügen Sie Regeln hinzu, damit bestimmter Datenverkehr Ihre Instance erreichen kann.

Sicherheitsgruppe erstellen Vorhandene Sicherheitsgruppe auswählen

Name der Sicherheitsgruppe - Pflichtfeld

docker-host-security-group

Diese Sicherheitsgruppe wird allen Netzwerkschnittstellen hinzugefügt. Der Name kann nicht bearbeitet werden, nachdem die Sicherheitsgruppe erstellt wurde. Die maximale Länge beträgt 255 Zeichen. Gültige Zeichen: a-z, A-Z, 0-9, Leerzeichen und _-:/() #,@[]+= &,:! \$*

Beschreibung - Pflichtfeld | [Info](#)

docker-host-security-group for docker-host EC2

Eingehende Sicherheitsgruppenregeln

▼ Sicherheitsgruppenregel 1 (TCP, 22, 213.47.200.121/32, SSH Rene) [Entfernen](#)

Typ Info	Protokoll Info	Portbereich Info
ssh	TCP	22
Quelltyp Info	Name Info	Beschreibung – optional Info
Meine IP	<input type="text"/> CIDR, Präfixliste oder Sicherheitsgruppe hinzufügen	SSH Rene
	<input type="text"/> 213.47.200.121/32	

16 GiB of disk space get used. It was deliberately chosen to be higher than the default of 8 GiB to have enough space for docker images.

▼ Speicher konfigurieren [Info](#) [Erweitert](#)

1x 15 GiB gp3 Stamm-Volume, 3000 IOPS, Unverschlüsselt

Kunden mit einem kostenlosen Kontingent können bis zu 30 GB EBS-Standardspeicher (SSD) oder Speicher auf Magnetfestplatten erhalten.

Neues Volume hinzufügen

Klicken Sie auf „Aktualisieren“, um die Backup-Informationen anzuzeigen.
Die von Ihnen zugewiesenen Tags bestimmen, ob die Instance durch irgendwelche Data Lifecycle Manager-Richtlinien gesichert wird.

0 x Dateisysteme [Bearbeiten](#)

After starting the instance, an elastic IP address is created and bound to the EC2 instance:

Elastische IP-Adresse zuordnen [Informationen](#)
Dieser elastischen IP-Adresse zuzuordnende Instance oder Netzwerkschnittstelle auswählen (98.83.248.165)

Elastische IP-Adresse: 98.83.248.165

Ressourcentyp
Wählen Sie den Ressourcentyp, mit dem die elastische IP-Adresse verknüpft werden soll.

Instance
 Netzwerkschnittstelle

⚠ Wenn Sie eine elastische IP-Adresse mit einer Instance verknüpfen, der bereits eine elastische IP-Adresse zugeordnet ist, wird die zuvor verknüpfte elastische IP-Adresse aufgehoben, die Adresse wird jedoch weiterhin Ihrem Konto zugewiesen. [Weitere Informationen](#)

Wenn keine private IP-Adresse angegeben ist, wird die elastische IP-Adresse mit der primären privaten IP-Adresse verknüpft.

Instance X C

Private IP-Adresse
Die private IP-Adresse, der die elastische IP-Adresse zugeordnet werden soll.

X

Neuzuordnung
Geben Sie an, ob die elastische IP-Adresse einer neuen Ressource zugeordnet werden kann, wenn sie bereits einer anderen Ressource zugeordnet ist.

Neuzuordnung dieser elastischen IP-Adresse zulassen

[Abbrechen](#) Zuordnen

The started instance is now reachable via SSH. It is important to mention that the file permissions of the previously created keypair in .pem file format must be changed to "700" for SSH to accept the key.

```
[rene@rene Downloads]$ chmod 700 main-keypair.pem
[rene@rene Downloads]$ ssh -i main-keypair.pem ec2-user@98.83.248.165
,#
~\_\_ ####_
~~ \_\#\#\#\\
~~ \#\#\|_
~~ \#/ _-- https://aws.amazon.com/linux/amazon-linux-2023
~~ V~' '-->
~~~ /
~~_. /_
~/_/
Last login: Mon Jun 9 13:18:19 2025 from 213.47.200.121
[ec2-user@ip-10-0-0-94 ~]$ █
```

3.3 Docker host configuration

First things first, docker is installed via “sudo yum install docker”. Afterwards the engine is started and the default user “ec2-user” added to the docker group. The installation can be verified by running a container of the hello-world image:

```
[ec2-user@ip-10-0-0-94 ~]$ sudo systemctl enable --now docker
[ec2-user@ip-10-0-0-94 ~]$ sudo usermod -aG docker $USER
[ec2-user@ip-10-0-0-94 ~]$ newgrp docker
[ec2-user@ip-10-0-0-94 ~]$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

[ec2-user@ip-10-0-0-94 ~]$ █
```

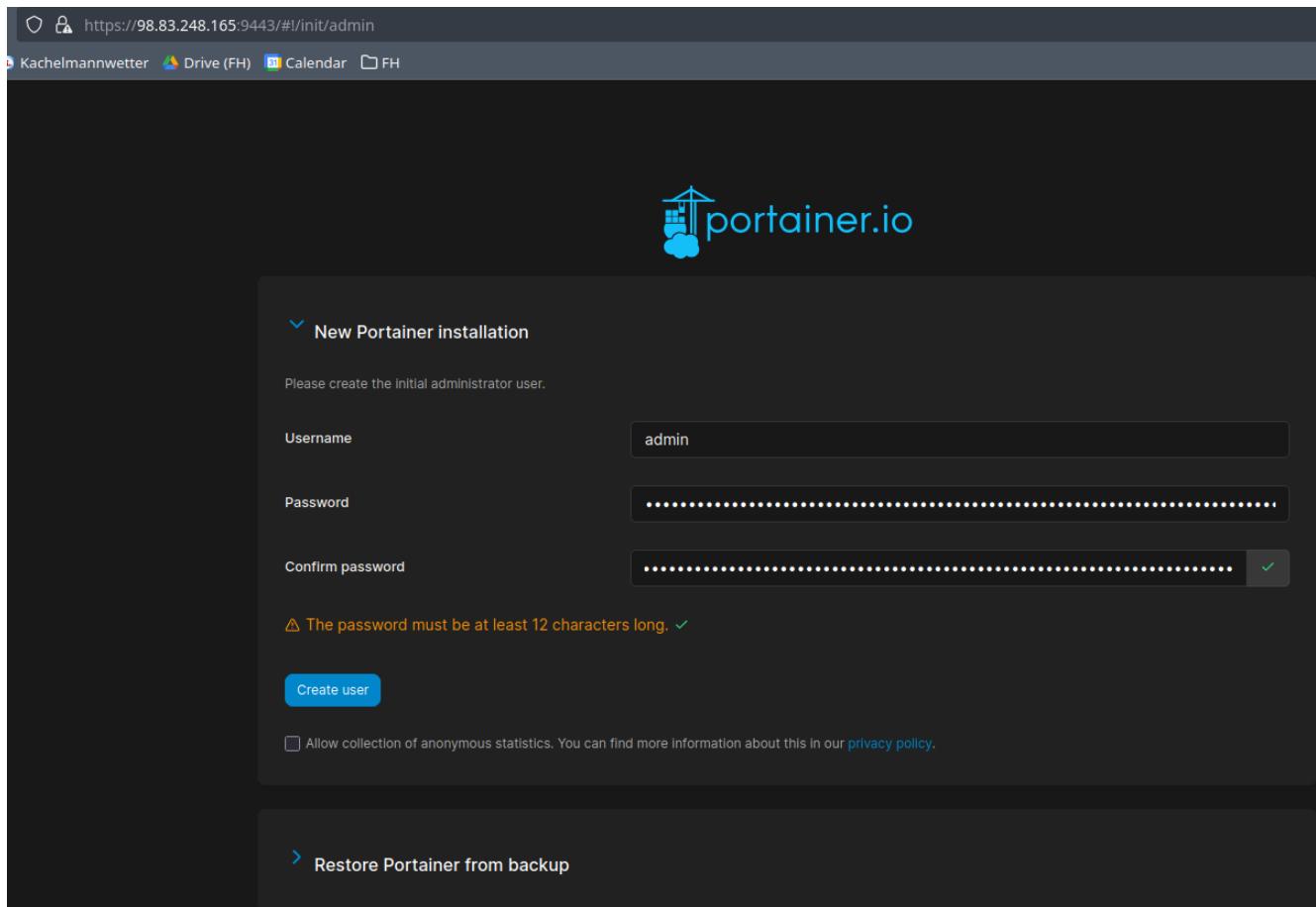
We also install docker compose:

```
[ec2-user@ip-10-0-0-94 deployment]$ sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
% Total    % Received % Xferd  Average Speed   Time   Time     Time  Current
          Dload  Upload   Total Spent  Left Speed
 0       0      0      0      0      0      0      0      0      0
 0       0      0      0      0      0      0      0      0      0
100  71.5M  100  71.5M    0      0  86.9M      0      0      86.9M
[ec2-user@ip-10-0-0-94 deployment]$
```

For graphical management of the docker environment on this server, we use Portainer that can be spun up as a container. First, a volume is created. Then the Portainer container can be created.

```
[ec2-user@ip-10-0-0-94 ~]$ docker volume create portainer_data
portainer_data
[ec2-user@ip-10-0-0-94 ~]$ docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:lt
Unable to find image 'portainer/portainer-ce:lt' locally
lt: Pulling from portainer/portainer-ce
c02951246260: Pull complete
b7c5ecc05946: Pull complete
04de093ad5ed: Pull complete
a9ff7abff372: Pull complete
e09df2601140: Pull complete
54675ba571ac: Pull complete
886c883b7538: Pull complete
93b4ed907f92: Pull complete
f126cb2940fd: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:ebead33595e425f88b1d02a74e4cc65a6d295e96c3643bb176dca7cb64bc36b0
Status: Downloaded newer image for portainer/portainer-ce:lt
c461956e68a8276b1b23fd2e8cc21422fab16e3315330dc6c972b2117662c70
[ec2-user@ip-10-0-0-94 ~]$
```

After creating an incoming rule in the security group of the EC2 instance, Portainer is reachable via the internet. Upon first connection, an admin password must be set.



3.4 NGINX Proxy manager

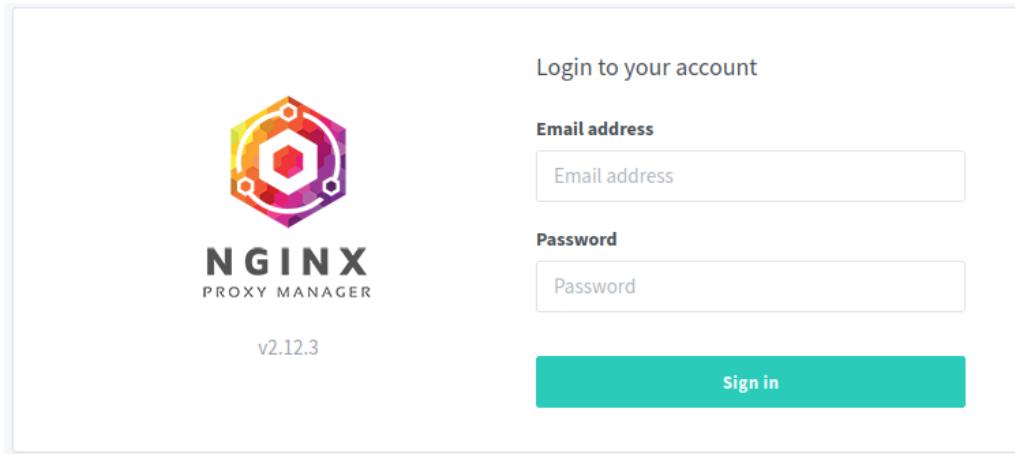
NGINX Proxy Manager will be used for managing TLS termination including certificates via Letsencrypt. A domain has been bought for this project: conint-securenotes.online. DNS is set up in a way, that all subdomains get routed to the AWS instance (98.83.248.165):

Type	Host	Value	TTL
A Record	*	98.83.248.165	Automatic

A new stack (docker compose stack) can be added in Portainer containing configuration for the NGINX proxy manager container which will use a custom docker compose in the repository:

```
services:
  nginx-proxy-manager:
    image: jc21/nginx-proxy-manager:2
    container_name: nginx-proxy-manager
    restart: unless-stopped
    ports:
      - "80:80"
      - "81:81"
      - "443:443"
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt
    networks:
      - jenkins_jenkins_network
      - portainer_portainer_network
      - sonarqube_sonarqube_network
      - proxy_network
```

Port 81 must be used for the initial setup. After adding an inbound rule in AWS for the administrator's IP only and port 81, NGINX proxy manager can be reached via internet. The initial credentials are "admin@example.com" with password "changeme".



A proxy host can be setup to move away from the insecure port 81.

Edit Proxy Host ×

[Details](#) [Custom locations](#) [SSL](#) [Advanced](#)

Domain Names *

reverse-proxy.conint-securenotes.online

Scheme * http **Forward Hostname / IP *** localhost **Forward Port *** 81

Cache Assets Block Common Exploits

Websockets Support

Access List

Publicly Accessible

[Cancel](#) [Save](#)

Edit Proxy Host

Details Custom locations SSL Advanced

SSL Certificate

reverse-proxy.conint-securenotes.online

Force SSL HTTP/2 Support

HSTS Enabled ? HSTS Subdomains

After this, port 81 can be removed again from the Portainer stack aswell as the inbound rule in AWS.

3.5 RDS setup

Creating a new RDS postgres database:

Datenbank erstellen Informationen

Auswahl einer Datenbank-Erstellungsmethode

Standard-Erstellung
Sie legen alle Konfigurationsoptionen fest, einschließlich der Optionen für Verfügbarkeit, Sicherheit, Sicherungen und Wartung.

Einfache Erstellung
Verwenden Sie empfohlene Best-Practice-Konfigurationen. Einige Konfigurationsoptionen können nach Erstellung der Datenbank geändert werden.

Engine-Optionen

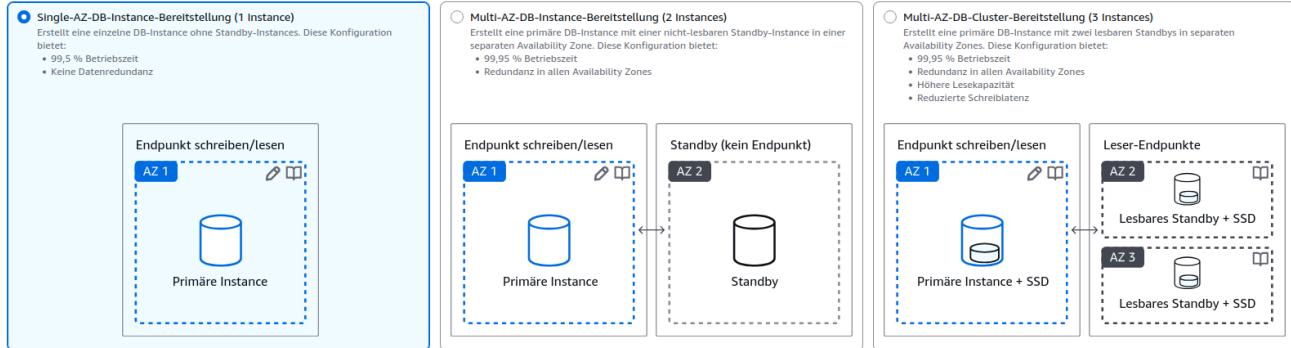
Engine-Typ <small>Informationen</small>	Icon	Description
<input type="radio"/> Aurora (MySQL Compatible)		
<input type="radio"/> Aurora (PostgreSQL Compatible)		
<input type="radio"/> MySQL		
<input checked="" type="radio"/> PostgreSQL		
<input type="radio"/> MariaDB		
<input type="radio"/> Oracle		ORACLE
<input type="radio"/> Microsoft SQL Server		
<input type="radio"/> IBM Db2		IBM Db2

Selecting a single AZ database instance:

Verfügbarkeit und Beständigkeit

Bereitstellungsoptionen [Informationen](#)

Wählen Sie die Bereitstellungsoption, die für Ihren Anwendungsfall erforderliche Verfügbarkeit und Langlebigkeit bietet. AWS verpflichtet sich zu einer bestimmten Betriebszeit, die von der gewählten Bereitstellungsoption abhängt. Weitere Informationen finden Sie unter [Amazon RDS Service Level Agreement \(SLA\)](#).



Using a generated password and not using AWS secrets manager as this would add cost:

Einstellungen

DB-Instance-Kennung [Informationen](#)

Geben Sie einen Namen für Ihre DB-Instance ein. Der Name muss für alle DB-Instances Ihres AWS-Kontos in der aktuellen AWS-Region eindeutig sein.

database-1

Für die DB-Instance-Kennung wird die Groß-/Kleinschreibung nicht berücksichtigt. Sie wird stattdessen vollständig in Kleinbuchstaben gespeichert (wie z. B. in „meinedbinstance“). Einschränkungen: Zwischen 1 und 63 alphanumerische Zeichen oder Bindestriche. Muss mit einem Buchstaben beginnen. Darf keine zwei aufeinanderfolgenden Bindestriche enthalten. Darf nicht mit einem Bindestrich enden.

▼ Einstellungen für Anmeldeinformationen

Hauptbenutzername [Informationen](#)

Geben Sie eine Anmelde-ID für den Hauptbenutzer Ihrer DB-Instance ein.

postgres

1 bis 16 alphanumerische Zeichen. Muss mit einem Buchstaben beginnen.

Verwaltung von Anmeldeinformationen

Sie können AWS Secrets Manager verwenden oder Ihre Hauptbenutzer-Anmeldeinformationen verwalten.

In AWS Secrets Manager verwaltet - **am sichersten**

RDS generiert ein Passwort für Sie und verwaltet es während seines gesamten Lebenszyklus mithilfe von AWS Secrets Manager.

Passwort automatisch generieren

Amazon RDS kann ein Passwort für Sie generieren oder Sie können Ihr eigenes Passwort angeben.

ⓘ Sie können Ihre Anmeldeinformationen einsehen, nachdem Sie Ihre Datenbank erstellt haben. Klicken Sie auf "Anmeldeinformationen anzeigen" im Banner für die Datenbankerstellung, um das Passwort anzuzeigen.

Establishing a connection to the docker host EC2 instance. We let AWS create a subnet for the database.

Anbindung [Informationen](#)



Datenverarbeitungsressource

Wählen Sie aus, ob eine Verbindung zu einer Datenverarbeitungsressource für diese Datenbank eingerichtet werden soll. Durch das Einrichten einer Verbindung werden die Konnektivitätseinstellungen automatisch geändert, sodass die Datenverarbeitungsressource eine Verbindung zu dieser Datenbank herstellen kann.

Keine Verbindung zu einer EC2-Datenverarbeitungsressource herstellen

Richten Sie keine Verbindung zu einer Datenverarbeitungsressource für diese Datenbank ein. Sie können später manuell eine Verbindung zu einer Datenverarbeitungsressource einrichten.

Herstellen einer Verbindung mit einer EC2-Datenverarbeitungsressource

Richten Sie eine Verbindung zu einer EC2-Datenverarbeitungsressource für diese Datenbank ein.

EC2-Instance [Informationen](#)

Wählen Sie die EC2-Instance aus, die als Datenverarbeitungsressource für diese Datenbank hinzugefügt werden soll. Dieser EC2-Instance wird eine VPC-Sicherheitsgruppe hinzugefügt. Außerdem wird der Datenbank eine VPC-Sicherheitsgruppe mit einer Regel für eingehenden Datenverkehr hinzugefügt, die der EC2-Instance den Zugriff auf die Datenbank erlaubt.

I-0d22fc0c2a2d2be6c
docker-host



ⓘ Einige VPC-Einstellungen können nicht geändert werden, wenn eine Datenverarbeitungsressource hinzugefügt wird

Durch das Hinzufügen einer EC2-Datenverarbeitungsressource werden automatisch die Einstellungen für die VPC, die DB-Subnetzgruppe und den öffentlichen Zugriff für diese Datenbank ausgewählt. Damit die EC2-Instance auf die Datenbank zugreifen kann, wird der Datenbank die VPC-Sicherheitsgruppe rds-ec2-X und der EC2-Instance eine weitere Sicherheitsgruppe mit dem Namen ec2-rds-X hinzugefügt. Sie können die neue Sicherheitsgruppe für die Datenbank nur entfernen, indem Sie die Datenverarbeitungsressource entfernen.

Virtual Private Cloud (VPC) [Informationen](#)

Wählen Sie die VPC aus. Die VPC definiert die virtuelle Netzwerkumgebung für diese DB-Instance.

conint-vpc (vpc-0bd7cd0e3b0a09cbe)

2 Subnetze, 2 Availability Zones

Es werden nur VPCs mit einer entsprechenden DB-Subnetzgruppe aufgelistet.

ⓘ Nachdem eine Datenbank erstellt wurde, können Sie Ihre VPC nicht mehr ändern.

DB-Subnetzgruppe [Informationen](#)

Wählen Sie die DB-Subnetzgruppe aus. Die DB-Subnetzgruppe, die definiert, welche Subnetze und IP-Bereiche die DB-Instance in der von Ihnen ausgewählten VPC verwenden kann.

Vorhandene auswählen

Vorhandene DB-Subnetzgruppe auswählen

Automatische Einrichtung

RDS erstellt eine neue Subnetzgruppe für Sie oder verwendet eine vorhandene Subnetzgruppe erneut.

DB-Subnetzgruppenname

rds-ec2-db-subnet-group-1

Neue DB-Subnetzgruppe erstellen.

Using the security group of the EC2 instance for the connection rules:

Öffentlicher Zugriff Informationen

Ja

RDS weist der Datenbank eine öffentliche IP-Adresse zu. Amazon-EC2-Instances und andere Ressourcen außerhalb der VPC können eine Verbindung zu Ihrer Datenbank herstellen. Ressourcen innerhalb der VPC können sich auch mit der Datenbank verbinden. Wählen Sie eine oder mehrere VPC-Sicherheitsgruppen aus, die angeben, welche Ressourcen eine Verbindung mit der Datenbank herstellen können.

Nein

RDS weist der Datenbank keine öffentliche IP-Adresse zu. Nur Amazon-EC2-Instances und andere Ressourcen innerhalb der VPC können eine Verbindung zu Ihrer Datenbank herstellen. Wählen Sie eine oder mehrere VPC-Sicherheitsgruppen aus, die angeben, welche Ressourcen eine Verbindung mit der Datenbank herstellen können.

VPC-Sicherheitsgruppe (Firewall) Informationen

Wählen Sie eine oder mehrere VPC-Sicherheitsgruppen aus, um den Zugriff auf Ihre Datenbank zu erlauben. Stellen Sie sicher, dass die Sicherheitsgruppenregeln den entsprechenden eingehenden Datenverkehr zulassen.

Vorhandene wählen

Bestehende VPC-Sicherheitsgruppen wählen

Neu erstellen

Neue VPC-Sicherheitsgruppe erstellen

Zusätzliche VPC-Sicherheitsgruppe

Eine oder mehrere Optionen auswählen

docker-host-security-group 

Amazon RDS fügt eine neue VPC-Sicherheitsgruppe hinzu, rds-ec2-1 um die Konnektivität mit Ihrer Datenverarbeitungsressource zu ermöglichen.

Availability Zone Informationen

us-east-1a

Zertifizierungsstelle - optional Informationen

Die Verwendung eines Serverzertifikats bietet eine zusätzliche Sicherheitsebene, indem überprüft wird, ob die Verbindung zu einer Amazon-Datenbank hergestellt wird. Dazu überprüfen Sie das Serverzertifikat, das automatisch auf allen von Ihnen bereitgestellten Datenbanken installiert wird.

rds-ca-rsa2048-g1 (Standard)

Ablauf: May 26, 2016

Wenn Sie keine Zertifizierungsstelle auswählen, wählt RDS eine für Sie aus.

Setting logging and monitoring to basic settings to save cost:

Überwachung Informationen

Wählen Sie Überwachungstools für diese Datenbank. Database Insights bietet eine kombinierte Ansicht von Performance Insights und Enhanced Monitoring für Ihre Datenbankflotte. Die Preise für Database Insights sind unabhängig von den monatlichen RDS-Schätzungen. Siehe [Amazon-CloudWatch-Preise](#).

Datenbanküberblicke – Fortgeschritten

- Speichert einen 15-monatigen Leistungsverlauf
- Überwachung auf Flottenebene
- Integration mit CloudWatch-Anwendungssignalen

Datenbanküberblicke – Standard

- Speichert den Leistungsverlauf von 7 Tagen, mit der Option, für die Beibehaltung des Leistungsverlaufs von bis zu 24 Monaten zu zahlen

Performance-Insights

Performance-Insights aktivieren

Mit dem Performance-Insights-Dashboard können Sie die Auslastung Ihrer Amazon-RDS-DB-Instance visualisieren und die Last nach Wartezeiten, SQL-Anweisungen, Hosts oder Benutzern filtern.

▼ Zusätzliche Überwachungseinstellungen

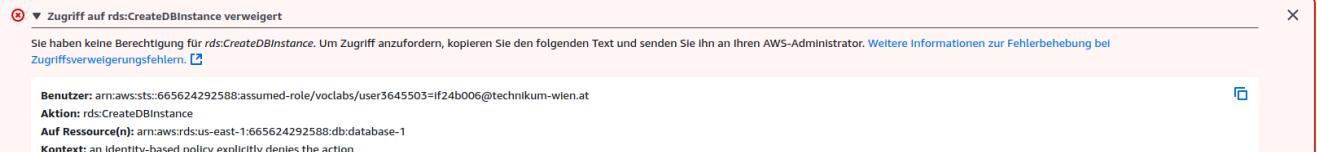
Verbesserte Überwachung, CloudWatch-Protokolle und DevOps Guru

Enhanced Monitoring

Erweiterte Überwachung aktivieren

Metriken für das Aktivieren der erweiterten Überwachung sind nützlich, um zu erkennen, wie unterschiedliche Prozesse oder Threads die CPU nutzen.

Unfortunately, an error occurs while trying to create a RDS instance:



The problem is that the lab user does not have sufficient permissions to use this AWS service. As a workaround, it was decided to host a Postgres instance on our Docker host. For this, a network called "deployment-network" was created. The proxy manager was added to this network. A new stack is set up running a Postgres instance:

```
services:
  db:
    image: postgres:17-alpine
    container_name: db
    restart: unless-stopped
    env_file:
      - docker.env
    expose:
      - 5432
    volumes:
      - secret_notes_postgres:/var/lib/postgresql/data
    networks:
      - secret_notes_network
```

4. Docker Hub

- Repository setup and naming conventions
- Image tagging strategy

Two publicly visible Docker Hub repositories are set up:

Name	Last Pushed	Contains	Visibility	Scout
nick7152/secret-notes-backend	36 minutes ago	IMAGE	Public	Inactive
nick7152/secret-notes-frontend	36 minutes ago	IMAGE	Public	Inactive

Both are named with the prefix "secret-notes-" and get discriminated by the layer (frontend / backend).

Images built in a CI pipeline will be tagged with the latest full commit SHA digest. The newest image will additionally carry the tag "latest". An example:

The screenshot shows the Docker Hub repository page for `nick7152/secret-notes-backend`. The repository was last pushed 37 minutes ago and has a size of 211.1 MB. The `Tags` tab is selected, showing two entries:

- latest**: Last pushed 37 minutes ago by `nick7152`. Digest: `29b6762bce58`, OS/ARCH: `linux/amd64`, Last pull: less than 1 day, Compressed size: 60.47 MB.
- 29b6762bce58**: Last pushed 37 minutes ago by `nick7152`. Digest: `2eff810b0dc42f2b0b5d5f364368be559d3fe18c`, OS/ARCH: `linux/amd64`, Last pull: less than 1 day, Compressed size: 60.47 MB.

Docker commands for pulling the latest image are displayed on the right:

```
docker pull nick7152/secret-notes-backend:latest
```

And for pulling the specific digest:

```
docker pull nick7152/secret-notes-backend:2eff810b0dc42f2b0b5d5f364368be559d3fe18c
```

Here the latest tag refers to the newest image "2eff810b0dc42f2b0b5d5f364368be559d3fe18c".

5. CI Servers

- Details of both pipelines (cloud-hosted and self-hosted)
- Screenshots or logs showing successful and failing runs

5.1 GitHub Actions

Enabling CI/CD functionality to a GitHub repository can be done by specifying workflow files in YAML format in the directory ".github/workflows" in the repository.

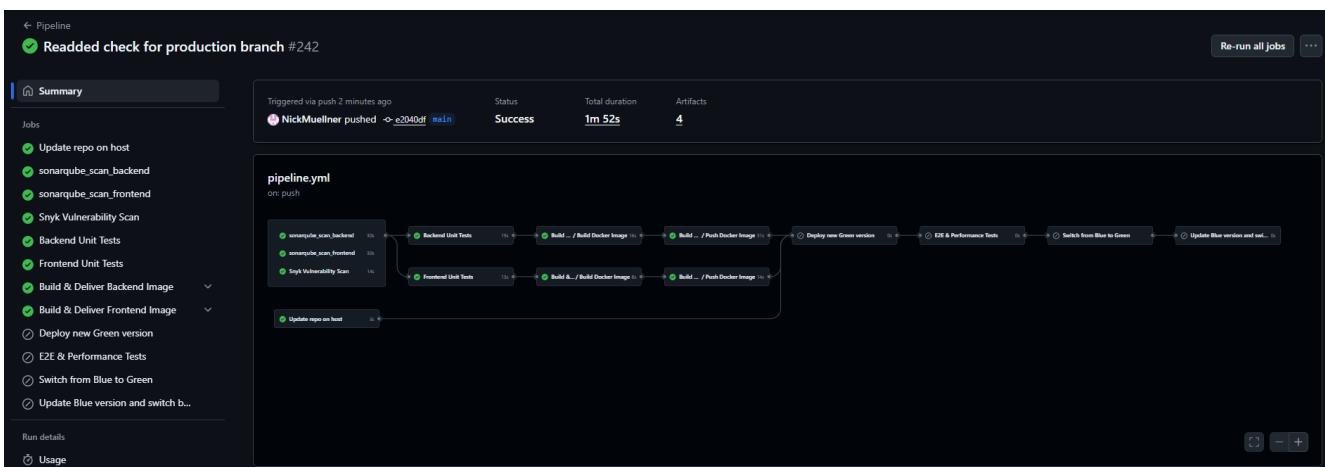
The main pipeline is defined as "pipeline.yml". The long steps of building and delivering docker images has been outsourced to a second workflow file "build-and-deliver.yml" that can be called in the main workflow multiple times. This mechanism can be compared to functions in code as we can pass input variables to the sub-workflow.

As GitHub does not support the concept of stages like GitLab does, stage-like behavior can be simulated using "needs" directives that control the flow. Additionally, if clauses can be used to determine if a job should run.

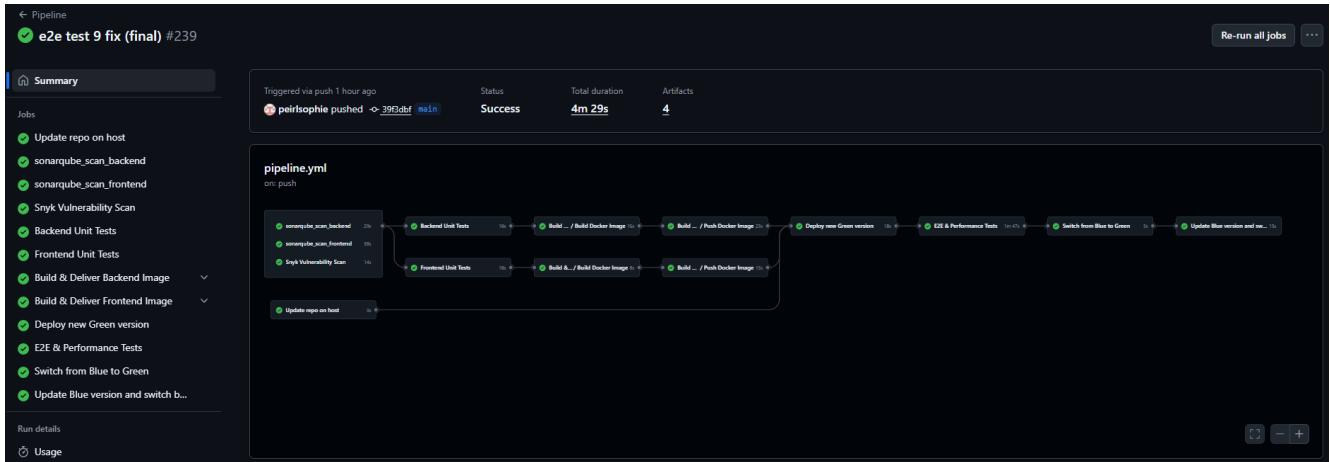
From left to right the following stages are contained:

- Linting: Contains SAST scanning with SonarQube, vulnerability scanning with Snyk and eslint
- Unit testing
- Build: Building Docker images
- Deliver: Pushing Docker images to Docker Hub
- Deploy: Uses the latest images and deploys to staging (green) environment
- E2E & Performance testing: conducting tests to assure stability of newly deployed staging environment (playwright, k6)
- Go-live: switch traffic to the verified green environment in the first step, update blue and switch back in the second step.

The stages Deliver, Deploy, E2E & Performance and Go-live only run when the triggering commit was made on the branch "production". This is controlled via if clauses on the relevant jobs. The first stages are run when a commit is made to the main branch:



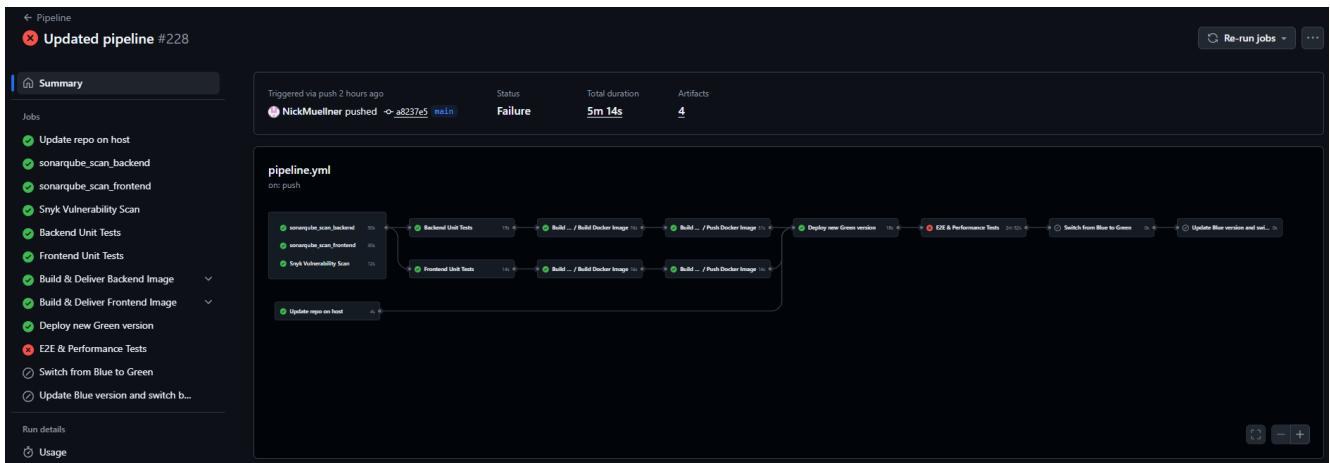
Without branch check for testing purposes:



The pipeline halts when a stage is unsuccessful. The following stages will not be executed in the event of failure.

Details of each stage are explained in chapter 9.

An example of a failed run:



E2E testing failed. This causes following stages to not be executed at all.

5.2 Jenkins

The setting with Jenkins is very similar to GitHub Actions. Instead of a pipeline.yml, a Jenkinsfile with groovy language is defined. But before we can start writing our stages, the setup for a Jenkins container is needed. Jenkins is available on Docker Hub as a container which we will use since we are using Docker on our host anyway. To address additional steps to have a container with Jenkins and project-related dependencies, a Dockerfile was created:

```
FROM jenkins/jenkins:lts

USER root

RUN groupadd -g 992 docker && usermod -aG docker jenkins

RUN gpg -k && gpg --no-default-keyring --keyring /usr/share/keyrings/k6-archive-keyring.gpg \
    --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys C5AD17C747E3415A3642D57D77C6C491D6AC1D69 && \
    echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" | tee /etc/apt/sources.list.d/k6.list

RUN apt-get update && \
    apt-get install -y k6

RUN curl -fsSL https://deb.nodesource.com/setup_22.x | bash - && \
    apt-get update && \
    apt-get install -y docker.io nodejs

RUN curl -L https://npmjs.org/install.sh | sh

RUN npx playwright install-deps

RUN curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-linux-x86_64" -o /usr/local/bin/docker-compose && \
    chmod +x /usr/local/bin/docker-compose

USER jenkins

RUN npx playwright install
```

This creates a Jenkins container with:

- Jenkins assigned to the Docker group (needed for Docker socket access)
- Installed k6, docker, docker-compose nodejs, npm and playwright packages

Portainer is again used to deploy the container on the host with this compose file:

```
services:
  jenkins:
    build:
      dockerfile: Dockerfile
    container_name: jenkins
    user: jenkins
    restart: unless-stopped
    expose:
      - 8080
    volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - jenkins_network

volumes:
  jenkins_home:

networks:
  jenkins_network:
```

As Jenkins is responsible for local rollouts and tests, the network Jenkins_home is added to most of the other containers.

Containers in network	
Container Name	IPv4 Address
backend-green	172.27.0.2/16
frontend-green	172.27.0.4/16
nginx-proxy-manager	172.27.0.6/16
sonarqube	172.27.0.5/16
jenkins	172.27.0.3/16

Now we can create a new Pipeline on the server with our Jenkinsfile with nearly 300 lines. Variables are either set or retrieved when starting a new run in the UI:

```
pipeline {
    agent any

    parameters {
        booleanParam(name: 'STATIC_TESTS', defaultValue: true, description: 'Statistische Code Analyse durchführen? (lint, snyk, sonar, jest)')
        booleanParam(name: 'DYNAMIC_TESTS', defaultValue: true, description: 'Dynamische Tests durchführen? (playwright, kō)')
        booleanParam(name: 'BUILD_FRONTEND', defaultValue: false, description: 'Frontend bauen und pushen?')
        booleanParam(name: 'BUILD_BACKEND', defaultValue: false, description: 'Backend bauen und pushen?')
        booleanParam(name: 'DEPLOY', defaultValue: false, description: 'Erzeugten Build deployen und testen?')
        string(name: 'IMAGE_TAG', defaultValue: 'latest', description: 'Tag unter dem das Docker image gepushed wird')
    }

    environment {
        SONAR_HOST_URL = 'http://sonarqube:9000'
        FRONTEND_GREEN = 'http://frontend-green:80'
        BACKEND_GREEN = 'http://backend-green:3000/api'
        FRONTEND_IMAGE = 'nick7152/secret-notes-frontend'
        BACKEND_IMAGE = 'nick7152/secret-notes-backend'
    }

    stages {
        stage('Install Dependencies') {
            parallel {
                stage('Install Frontend Dependencies') {
                    steps {
                        dir('frontend') {
                            echo 'Installing frontend dependencies...'
                            sh 'npm ci'
                        }
                    }
                }
            }
        }
    }
}
```

Jenkins comes with handy plugins for everything, but since we installed nearly everything on the host (e.g. nodejs, npm, etc.) we just added some graphical plugins and a plugin for SonarQube to the already installed default plugins:

- Blue Ocean → Huge overall UI enhancement
- Pipeline: Stage View Plugin → Summary of the last runs with stage detail
- Rebuilder → Easily rebuild a run with the same parameters
- SonarQube Scanner for Jenkins → Retrieves SonarQube scan results

Additional setup is needed for SonarQube:

SonarQube servers

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

Environment variables

SonarQube Installationen

Liste der SonarQube Installationen

Name
SonarQube

Server URL

Voreinstellung ist <http://localhost:9000>

<https://sonar.conint-securenotes.online/>

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

SONAR_TOKEN

+ Add

Erweitert ▾

Also, credentials are needed for the Pipeline:

Globale Zugangsdaten

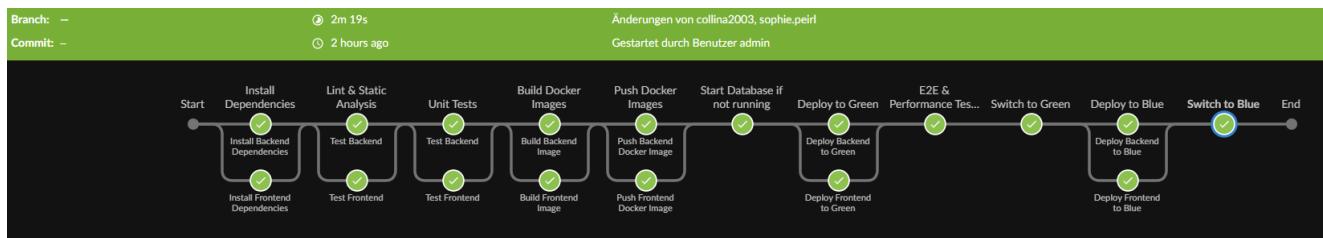
+ Add Credentials

Alle Zugangsdaten, die nicht zu einer bestimmten Domäne gehören.

ID	Name	Art	Beschreibung
sonar-creds	SONAR_TOKEN	Secret text	SONAR_TOKEN
snyk-creds	SNYK_TOKEN	Secret text	SNYK_TOKEN
dockerhub-creds	nick7152/***** (DOCKER_USER,DOCKER_PASS)	Benutzername und Passwort	DOCKER_USER,DOCKER_PASS

With everything set up, the pipeline can be manually triggered. To see a summary of some of the last runs we can easily look into the Pipeline and take use of the Pipeline: Stage View Plugin:

Here we see the last 10 runs and also which of them were successful. Because of the usage of parallel stages, unfortunately this view is not good for a detailed view of the stages. However, it is very nice to get a quick overview of the past runs. Therefore, we have Blue Ocean where we can inspect a specific run and also see parallel stages:



The following stages are run:

- Install dependencies
 - Lint & Static Analysis
 - Eslint
 - SonarQube
 - Snyk
 - Unit Tests
 - Build Docker Images
 - Push Docker Images
 - Start Database if not running (for first deployment)
 - Deploy to Green
 - E2E & Performance Tests
 - Switch to Green
 - Deploy to Blue
 - Switch to Blue

If a new run of the pipeline is started in the UI, several variables can be passed:

Input required

Statische Code Analyse durchführen? (lint, snyk, sonar, jest)

STATIC_TESTS

Dynamische Tests durchführen? (playwright, k6)

DYNAMIC_TESTS

Frontend bauen und pushen?

BUILD_FRONTEND

Backend bauen und pushen?

BUILD_BACKEND

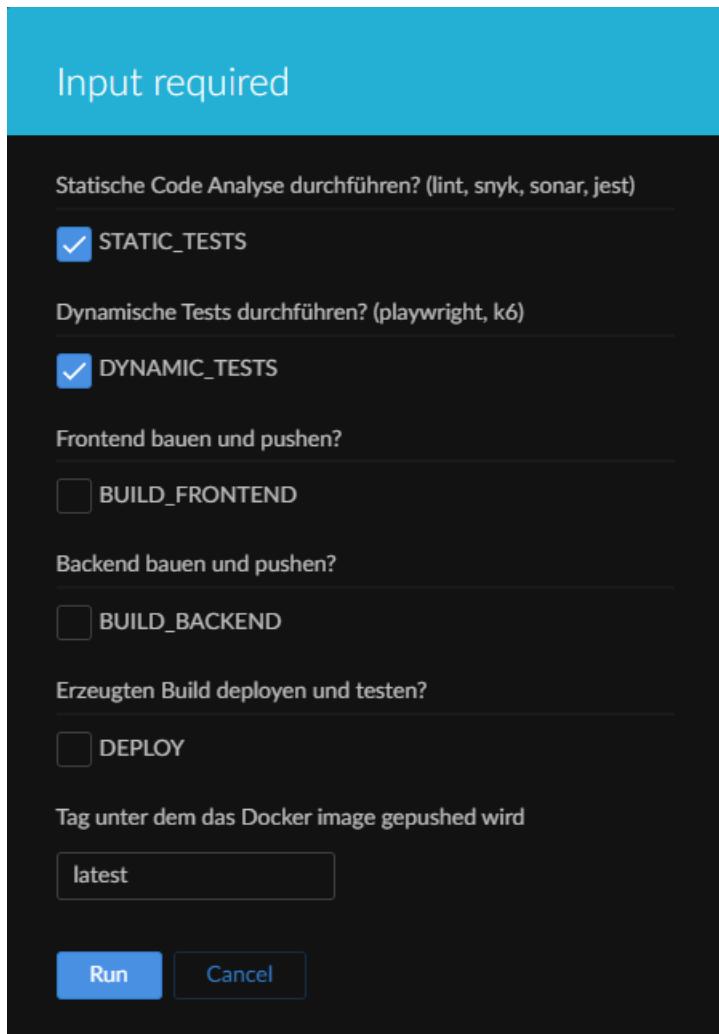
Erzeugten Build deployen und testen?

DEPLOY

Tag unter dem das Docker image gepushed wird

latest

Run **Cancel**



They have the following effects:

- STATIC_TESTS (default: true) → Executes stage “Lint & Static Analysis” and “Unit Tests”
- DYNAMIC_TESTS (default: true) → Executes stage “E2E & Performance Testing”
- BUILD_FRONTEND (default: false) → Executes stage “Build Docker Images: Build Frontend Image” and “Push Docker Images: Push Frontend Docker Image”
- BUILD_BACKEND (default: false) → Executes stage “Build Docker Images: Build Backend Image” and “Push Docker Images: Push Backend Docker Image”
- DEPLOY (default: false) → Executes stage “Start Database if not running”, “Deploy to Green”, “Switch to Green”, “Deploy to Blue” and “Switch to Blue”
- IMAGE_TAG (default: “latest”) → Defines which Image tag should be build/pushed and deployed

6. Code Quality Server

- SonarQube setup, including typical metrics collected

6.1 SonarQube container setup

A SonarQube server will be spun up on the EC2 docker host instance as a docker container. A problem occurs where the max map count of the system is set too low. The output of the error is visible when starting the SonarQube container. To fix, the file “/etc/sysctl.conf” needs to be edited with the needed map count. The configuration can be applied via “sudo sysctl –p”.

```
[ec2-user@ip-10-0-0-94 ~]$ cat /etc/sysctl.conf
# sysctl settings are defined through files in
# /usr/lib/sysctl.d/, /run/sysctl.d/, and /etc/sysctl.d/.
#
# Vendors settings live in /usr/lib/sysctl.d/.
# To override a whole file, create a new file with the same in
# /etc/sysctl.d/ and put new settings there. To override
# only specific settings, add a file with a lexically later
# name in /etc/sysctl.d/ and put new settings there.
#
# For more information, see sysctl.conf(5) and sysctl.d(5).
vm.max_map_count=262144
[ec2-user@ip-10-0-0-94 ~]$ sudo sysctl -p
vm.max_map_count = 262144
[ec2-user@ip-10-0-0-94 ~]$ █
```

A docker compose stack can be set up in Portainer. It uses SonarQube itself and a Postgres database for storing data. Sonar and Postgres need a couple of volumes to keep configurations and data persisted. An external network is created and used in both containers:

```
sonarqube:
  image: sonarqube:25.7.0.110598-community
  container_name: sonarqube
  restart: unless-stopped
  depends_on:
    - db
  expose:
    - 9000
  environment:
    SONAR_JDBC_URL: jdbc:postgresql://sonarqube_db:5432/sonarqube
    SONAR_JDBC_USERNAME: sonar
    SONAR_JDBC_PASSWORD: sonarpassword
  volumes:
    - sonarqube_data:/opt/sonarqube/data
    - sonarqube_extensions:/opt/sonarqube/extensions
    - sonarqube_logs:/opt/sonarqube/logs
    - sonarqube_temp:/opt/sonarqube/temp
  networks:
    - sonarqube_network
    - jenkins_jenkins_network
```

And the database:

```
db:
  image: postgres:17-alpine
  container_name: sonarqube_db
  restart: unless-stopped
  environment:
    POSTGRES_USER: sonar
    POSTGRES_PASSWORD: sonarpassword
    POSTGRES_DB: sonarqube
  volumes:
    - postgresql_data:/var/lib/postgresql/data
  networks:
    - sonarqube_network
```

After saving, the containers are up and running.

Containers								
Name	State	Quick Actions	Stack	Image	Created	IP Address	Published Ports	Ownership
sonar-db	healthy	Start Stop Kill Restart Pause Resume Remove	sonarqube	postgres:17	2025-06-12 13:17:56	172.21.0.2	-	administrators
sonarqube	running	Start Stop Kill Restart Pause Resume Remove	sonarqube	sonarqube:community	2025-06-12 13:17:56	172.21.0.3	-	administrators

A new proxy host is also added in NGINX proxy manager for the sonar instance (for TLS termination). The forward hostname can be specified as follows: container.network; in this case: "sonarqube.sonar-network". A SSL certificate is also requested via LetsEncrypt.

Edit Proxy Host

Details Custom locations SSL Advanced

Domain Names *

Scheme * Forward Hostname / IP * Forward Port *

http	sonarqube.sonar-networ	9000
------	------------------------	------

Cache Assets Block Common Exploits

Websockets Support

Access List

Publicly Accessible

Cancel Save

Sonar's interface is now accessible via <https://sonar.conint-securenotes.online> (a rule for incoming traffic has already been setup). Upon first login with credentials "admin" and password "admin", a new password must be entered.

The screenshot shows a web page titled "Update your password". At the top, there is a warning message: "⚠ This account should not use the default password." Below this, there are three input fields: "Old Password *", "Password *", and "Confirm Password *". Each field has a placeholder of five asterisks. A blue "Update" button is located at the bottom right of the form area.

Update your password

⚠ This account should not use the default password.

Enter a new password

Old Password *

Password *

Confirm Password *

Update

6.2 SAST configuration

Two new “local project” are configured in the Sonar interface, one for the backend and one for the frontend:

The screenshot shows the "Create your project" interface in the SonarQube Community application. It features a header with navigation links like Mail, YouTube, ORF, Kachelmannwetter, Drive (FH), Calendar, and FH. Below the header, there's a title "How do you want to create your project?". It asks if the user wants to benefit from SonarQube Community Build's features (repository import and Pull Request decoration). It then lists several import options: "Import from Azure DevOps", "Import from Bitbucket Cloud", "Import from Bitbucket Server", "Import from GitHub", and "Import from GitLab", each with a "Setup" button. At the bottom, there's a question "Are you just testing or have an advanced use-case?" followed by a "Create a local project" button.

How do you want to create your project?

Do you want to benefit from all of SonarQube Community Build's features (like repository import and Pull Request decoration)?

Create your project from your favorite DevOps platform.

First, you need to set up a DevOps platform configuration.

Import from Azure DevOps Import from Bitbucket Cloud Import from Bitbucket Server

Import from GitHub Import from GitLab

Are you just testing or have an advanced use-case?

Create a local project

1 of 2

Create a local project

Project display name* ⓘ

conint-securenotes

Project key* ⓘ

conint-securenotes

Main branch name*

main

The name of your project's default branch [Learn More](#) ⓘ

[Cancel](#)

[Next](#)

2 of 2

X

Set up project for Clean as You Code

The new code definition sets which part of your code will be considered new code. This helps you focus attention on the most recent changes to your project, enabling you to follow the Clean as You Code methodology. Learn more: [Defining New Code](#) ⓘ

Choose the baseline for new code for this project

Use the global setting

[Previous version](#)

Any code that has changed since the previous version is considered new code.

Recommended for projects following regular versions or releases.

Define a specific setting for this project

Previous version

Any code that has changed since the previous version is considered new code.

Recommended for projects following regular versions or releases.

Number of days

Any code that has changed in the last x days is considered new code. If no action is taken on a new issue after x days, this issue will become part of the overall code.

Recommended for projects following continuous delivery.

Reference branch

Choose a branch as the baseline for the new code.

Recommended for projects using feature branches.

[Back](#)

[Create project](#)

"GitHub actions" is selected for setup. Sonar provides detailed directions on how to add needed config to a GitHub repo. Firstly, a project token must be generated.

Analyze your project with GitHub CI

1 Create GitHub Secrets

In your GitHub repository, go to **Settings > Secrets and variables > Actions**

- 1 Click on **New repository secret**.
- 2 In the **Name** field, enter `SONAR_TOKEN` 
- 3 In the **Value** field, enter an existing token, or a newly generated one.
- 4 Click on **Add secret**.

- 1 Click on **New repository secret**.
- 2 In the **Name** field, enter `SONAR_HOST_URL` 
- 3 In the **Value** field, enter `https://98.83.248.165:9000` 
- 4 Click on **Add secret**.

Generate a project token

The project token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your [user account](#).

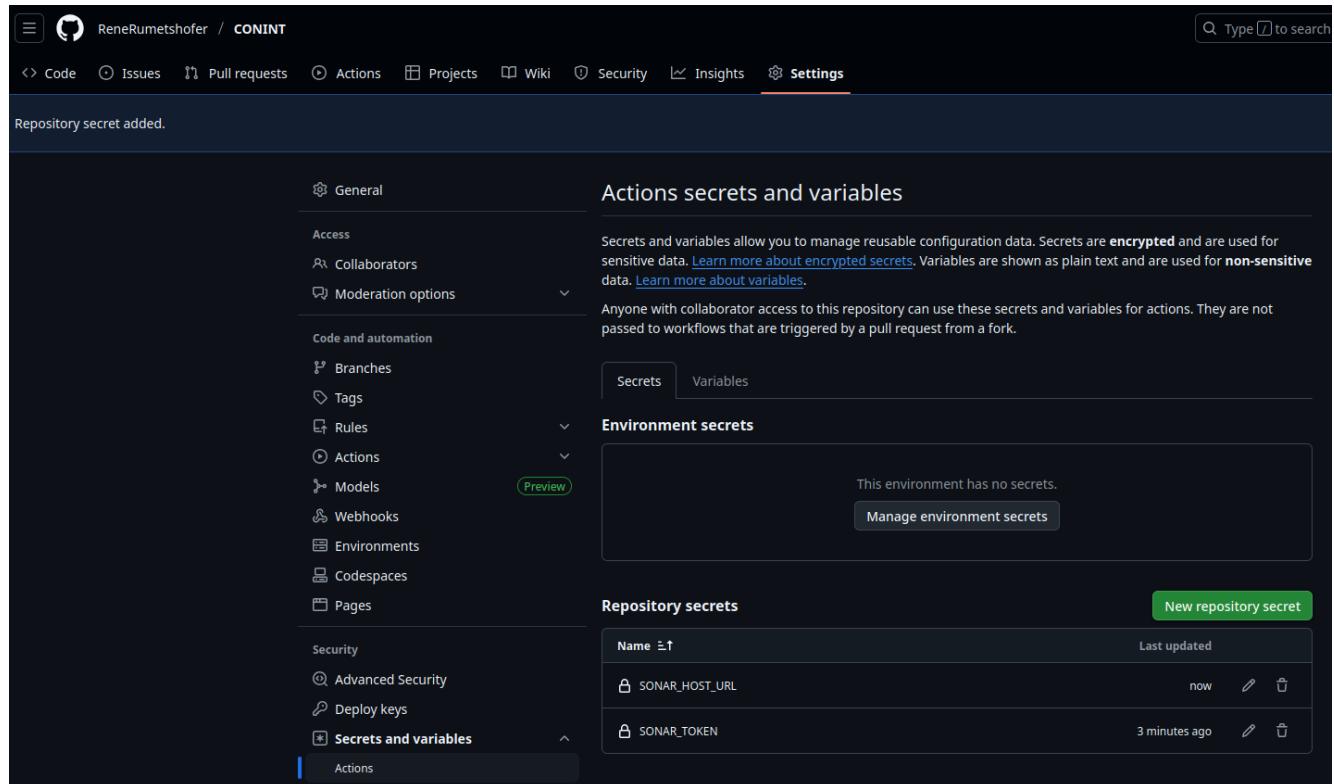
Analyze "conint-securenotes":

`sqp_8b5e1b7616498c33168ede8a9f71428a10f74e6c`  

 New token "`sqp_8b5e1b7616498c33168ede8a9f71428a10f74e6c`" has been created. Make sure you copy it now, you won't be able to see it again!

Continue

This token is then entered in the GitHub repo's settings under "Secrets and variables" -> "Actions" as a repository secret called "SONAR_TOKEN". Another secret "SONAR_HOST_URL" has to be created that contains the URL of Sonar.



Repository secret added.

Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. [Learn more about encrypted secrets](#). Variables are shown as plain text and are used for non-sensitive data. [Learn more about variables](#).

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Repository secrets

Name	Last updated
<code>SONAR_HOST_URL</code>	now
<code>SONAR_TOKEN</code>	3 minutes ago

New repository secret

Next, a file called "sonar-project.properties" with content "sonar.projectKey=conint-securenotes" is added to the root of the repository and committed. A test workflow is added under ".github/workflows/sonar.yml" for trying sonar connectivity:

```
sonarqube_scan_backend:  
  runs-on: ubuntu-latest  
  steps:  
    - uses: actions/checkout@v4  
    - uses: SonarSource/sonarqube-scan-action@v5  
      with:  
        args: -Dsonar.projectBaseDir=backend  
        env:  
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}  
          SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}  
    - uses: SonarSource/sonarqube-quality-gate-action@v1  
      with:  
        scanMetadataReportFile: backend/.scannerwork/report-task.txt  
        timeout-minutes: 5  
        env:  
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}  
  
sonarqube_scan_frontend:  
  runs-on: ubuntu-latest  
  steps:  
    - uses: actions/checkout@v4  
    - uses: SonarSource/sonarqube-scan-action@v5  
      with:  
        args: -Dsonar.projectBaseDir=frontend  
        env:  
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}  
          SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}  
    - uses: SonarSource/sonarqube-quality-gate-action@v1  
      with:  
        scanMetadataReportFile: frontend/.scannerwork/report-task.txt  
        timeout-minutes: 5  
        env:  
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```

After committing the workflow a pipeline runs through and conducts a check via our Sonar server:

Sonar check succeeded 20 minutes ago in 53s

- > Set up job 2s
- > Run actions/checkout@v4 1s
- > Run SonarSource/sonarqube-scan-action@v5 32s
- > Run SonarSource/sonarqube-quality-gate-action@v1 11s
 - 1 ► Run SonarSource/sonarqube-quality-gate-action@v1
 - 6 ► Run \$GITHUB_ACTION_PATH/script/check-quality-gate.sh scannerwork/report-task.txt
 - 11 ..
 - 12 ✓ Quality Gate has PASSED.
- > Post Run SonarSource/sonarqube-scan-action@v5 3s
- > Post Run actions/checkout@v4 0s
- > Complete job 0s

https://sonar.conint-securenotes.online/dashboard?id=conint-securenotes&codeScope=overall

SonarQube community Projects Issues Rules Quality Profiles Quality Gates Administration More

conint-securenotes / main ✓ ?

Overview Issues Security Hotspots Code Measures Activity Project Settings Project Information

main 45k Lines of Code Version not provided Set as homepage

Don't let issues accumulate. Discover 'Clean as You Code'! Learn how to improve your code base by cleaning only new code.

Take the Tour Not now

Quality Gate Passed Last analysis 21 minutes ago

New Code	Overall Code	
Security 1 Open issues E	Reliability 1 Open issues C	Maintainability 41 Open issues A
Accepted issues 0	Coverage 0.0% On 2.3k lines to cover.	Duplications 0.0% On 49k lines.
Security Hotspots 6 E		

In Jenkins we also need preparation:

1. Install and configure associated plugin as mentioned in 5.2
2. Configure credentials (same token as used in GitHub Actions)
3. Set up webhook in SonarQube to let Jenkins know when it has finished

Webhooks

Webhooks are used to notify external services when a project analysis is done. An HTTP POST request including a JSON payload is sent to each of the provided URLs. Learn more in the [Webhooks documentation](#).

Name	URL	Has secret?	Last delivery	Actions
Jenkins	http://jenkins.conint-securenotes.online/sonarqube-webhook/	No	July 14, 2025 at 12:30 AM	⋮

Now Jenkins executes the scan and aborts if there was found something:

```
echo 'SonarQube & Snyk Frontend...'
withCredentials([string(credentialsId: 'sonar-creds', variable: 'SONAR_TOKEN')]) {
    withSonarQubeEnv('SonarQube') {
        sh 'npm run sonar-scanner'
    }
}
timeout(time: 1, unit: 'MINUTES') {
    waitForQualityGate abortPipeline: true
}
```

```
true – Wait for SonarQube analysis to be completed and return quality gate status
1 Checking status of SonarQube task '9e55994b-ebe3-4f28-84a6-d99418ecad44' on server 'SonarQube'
2 SonarQube task '9e55994b-ebe3-4f28-84a6-d99418ecad44' status is 'IN_PROGRESS'
3 SonarQube task '9e55994b-ebe3-4f28-84a6-d99418ecad44' status is 'SUCCESS'
4 SonarQube task '9e55994b-ebe3-4f28-84a6-d99418ecad44' completed. Quality gate is 'OK'
```

7. Security Scan Server

- Snyk usage, how to interpret and fix vulnerabilities found

On Snyk a new organization “CONINT-securenotes” is used, the ID can be obtained via choosing to add a new project via “CLI”. For authentication, a token will be used. This can be obtained in the personal settings of Snyk in the “General” section.

Both the organization ID and the authentication token are stored as repository secrets in GitHub:

Repository secrets		New repository secret
Name	Last updated	
SNYK_ORG	47 minutes ago	
SNYK_TOKEN	last week	

A new job is added to the GitHub workflow:

```
vulnerability_scan:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@main
    - uses: snyk/actions/setup@master
    - uses: actions/setup-node@v4
    - name: Snyk monitor
      run: snyk monitor --all-projects --org=${{ secrets.SNYK_ORG }}
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
    - name: Snyk test
      run: snyk test --all-projects --org=${{ secrets.SNYK_ORG }}
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
```

No dependencies are specified as the job should run in the first “stage”. It was decided to use the monitoring function to show vulnerabilities in Snyk’s dashboard and to get notified via email when new vulnerabilities should occur. The problem with this is that the pipeline does not fail if vulnerabilities are discovered. Therefore, we opted to run a Snyk test as well.

Running the pipeline for the first time resulted in a finding:

```
vulnerability_scan
failed 1 minute ago in 14s

▼ ✓ Snyk monitor
▼ ✘ Snyk test

1 ► Run snyk test --all-projects --org=***
2
3 Testing /home/runner/work/CONINT/CONINT...
4
5 Tested 100 dependencies for known issues, found 2 issues, 2 vulnerable paths.
6
7
8 Issues to fix by upgrading:
9 Upgrade fastify@5.2.2 to fastify@5.3.2 to fix
10 ✘ Improper Validation of Specified Type of Input [High Severity][https://security.snyk.io/vuln/SNYK-JS-FASTIFY-9788069] in fastify@5.2.2
11 introduced by fastify@5.2.2
12
13 Issues with no direct upgrade or patch:
14 ✘ Regular Expression Denial of Service (ReDoS) [Low Severity][https://security.snyk.io/vuln/SNYK-JS-BRACEEXPANSION-9789073] in brace-expansion@2.0.1
15 introduced by postgrator@8.0.0 > glob@11.0.1 > minimatch@10.0.1 > brace-expansion@2.0.1
16 This issue was fixed in versions: 1.1.12, 2.0.2, 3.0.1, 4.0.1
17 Organization: conint-securenotes
18 Package manager: npm
19 Target file: backend/package-lock.json
20 Project name: securenotes-backend
21 Open source: no
22 Project path: /home/runner/work/CONINT/CONINT
23 Licenses: enabled
24 -----
25
26 Testing /home/runner/work/CONINT/CONINT...
27
28 Organization: conint-securenotes
29 Package manager: npm
30 Target file: frontend/package-lock.json
31 Project name: secret-notes-frontend
32 Open source: no
33 Project path: /home/runner/work/CONINT/CONINT
34 Licenses: enabled
35
36 ✅ Tested /home/runner/work/CONINT/CONINT for known issues, no vulnerable paths found.
37
38
39 Tested 2 projects, 1 contained vulnerable paths.
40
41
42 Error: Process completed with exit code 1.
```

René Rumetshofer/CONINT

Project	Imported	Tested	Issues ↓
securenotes-backend	an hour ago	2 minutes ago	0 C 1 H 0 M 1 L ...
secret-notes-frontend	an hour ago	2 minutes ago	0 C 0 H 0 M 0 L ...

In order to fix the high severity issue, an update of NPM packages was done by running "npm update".

After committing the change, the new pipeline passes the scan:

Summary

vulnerability_scan succeeded now in 21s

Search logs

Jobs

- sast (red)
- vulnerability_scan (green)
- test-backend (green)

Run details

Usage

Workflow file

```

vulnerability_scan
succeeded now in 21s

> ✓ Run actions/setup-node@v4
0s

> ✓ Snyk monitor
3s

✓ Snyk test
3s

1  ➔ Run snyk test --all-projects --org:***+
2
3
4  Testing /home/runner/work/CONINT/CONINT...
5
6  Organization: conint-securenotes
7  Package manager: npm
8  Target file: backend/package-lock.json
9  Project name: securenotes-backend
10 Open source: no
11 Project path: /home/runner/work/CONINT/CONINT
12 Licenses: enabled
13
14
15 ✓ Tested 101 dependencies for known issues, no vulnerable paths found.
16
17
18 -----
19
20 -----
21
22 Testing /home/runner/work/CONINT/CONINT...
23
24 Organization: conint-securenotes
25 Package manager: npm
26 Target file: frontend/package-lock.json
27 Project name: secret-notes-frontend
28 Open source: no
29 Project path: /home/runner/work/CONINT/CONINT
30 Licenses: enabled
31
32 ✓ Tested /home/runner/work/CONINT/CONINT for known issues, no vulnerable paths found.
33
34
35 Tested 2 projects, no vulnerable paths were found.

```

ReneRumetschofer/CONINT

Delete selected projects

Project

Project	Imported	Tested	Issues ↓
secret-notes-frontend	an hour ago	a few seconds ago	0 C 0 H 0 M 0 L ...
securenotes-backend	an hour ago	a few seconds ago	0 C 0 H 0 M 0 L ...

8. Feature Toggle & A/B Test Server

- PostHog setup for toggles and experiments
- Example toggles with instructions on how to enable/disable them

PostHog setup for toggles and experiments

The screenshot shows the PostHog interface for managing feature flags. The top navigation bar includes 'Default project', 'Feature Flags', and other project management options. The main page displays a feature flag named 'new-ui-theme'.

Key: new-ui-theme

Description (optional):

Overview (selected), **Usage**, **Projects**, **Schedule**, **History**, **Permissions**

STATUS: Enabled (switch is on)

TYPE: Multiple variants with rollout percentages (A/B/n test)

FLAG PERSISTENCE: This flag **does not persist** across authentication events.

Variant keys:

Key	Description	Payload	Rollout
A control	Blue Button	No payload associated with this variant	50% (View recordings)
B variant	Green Button	No payload associated with this variant	50% (View recordings)

Release conditions:

- Set 1 Condition set will match **all users**
- Rolled out to **100% of users in this set.**
- All **users** in this set have no variant override

Insights that use this feature flag:

You have no insights that use this feature flag. Explore this feature flag's insights by creating one below. Create insight

Loading PostHogs tracking library:

```
s.api_host.replace( searchValue: ".i.posthog.com", replaceValue: "-assets.i.posthog.com" ) + "/static/array.js" ).
```

Initialzing PostHog:

```
posthog.init( i: "phc_ASQ4jxULA4BuJQfLNnCH974PkeqxcLGfzQl2DyYXCjh", s: {
    api_host: "https://eu.i.posthog.com",
    defaults: "2025-05-24",
    person_profiles: "always",
});
```

Creating stub functions to allow calling PostHog methods before the library loads

```
function g(t, e) : void { Show usages & Nick Müllner
    var o = e.split(".");
    (2 == o.length && ((t = t[o[0]]), (e = o[1])));
    (t[e] = function () : void {
        t.push([e].concat(Array.prototype.slice.call(arguments, argArray: 0)));
    });
}
```

Example toggles with instructions on how to enable/disable them

In PostHogs' feature flags interface, you can toggle the feature flag with ease by pushing the button underneath status showcasing its current status. An advantage PostHog offers is that you don't have to restart the server when toggling the feature flag to apply its effects.

The screenshot below shows a PostHog feature flag to toggle the UI style of the create note button.

```
if (typeof posthog !== "undefined") {
    /* eslint-disable no-undef */
    posthog.onFeatureFlags(() : void => {
        const variant = posthog.getFeatureFlag("new-ui-theme");
        /* eslint-enable no-undef */
        const button : HTMLElement = document.getElementById(elementId: "createNote");
        if (button) {
            button.style.backgroundColor = variant === "variant" ? "green" : "blue";
        }
    });
}
```

9. Each Stage of the Pipeline

- Detailed explanation of Lint, Test, Build, Deliver, Deploy, E2E/Performance Testing
- What happens if a stage fails (notification)

9.1 GitHub actions

The main pipeline is defined as “pipeline.yml”. As mentioned in chapter 5, stages are not directly supported by GitHub actions. They are however simulated by controlling the flow of jobs via defining needed jobs with the “needs” keyword.

Before running the script, a git pull is executed on the EC2 host, because some files are needed locally like the nginx.conf for the blue-green-proxy or the blue/green compose files:

```
update-local-repo:
  name: Update repo on host
  runs-on: ubuntu-latest
  steps:
    - uses: appleboy/ssh-action@v1
      with:
        host: ${{ secrets.DEPLOY_HOST }}
        username: ${{ secrets.DEPLOY_HOST_USERNAME }}
        key: ${{ secrets.DEPLOY_HOST_SSH_KEY }}
        script: |
          cd /opt/CONINT && \
          sudo git pull
```

The linting stage runs two jobs: “sonarqube_scan_backend/sonarqube_scan_frontend” and “vulnerability_scan”. Existing actions are used for running SonarQube (SAST) and Snyk tests. The latter runs two snyk commands separately. The first “monitor” command only updates Snyk’s state of the repository (to manage in Snyk’s dashboard and get notified on new vulnerabilities) and does not fail the pipeline. This is why the consequent “snyk test” command is also used.

```
vulnerability_scan:
  name: Snyk Vulnerability Scan
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@main
    - uses: snyk/actions/setup@master
    - uses: actions/setup-node@v4
    - name: Snyk monitor
      run: snyk monitor --all-projects --org=${{ secrets.SNYK_ORG }}
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
    - name: Snyk test
      run: snyk test --all-projects --org=${{ secrets.SNYK_ORG }}
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
```

And SonarQube (first scanning and then retrieving results):

```
sonarqube_scan_backend:  
  runs-on: ubuntu-latest  
  steps:  
    - uses: actions/checkout@v4  
    - uses: SonarSource/sonarqube-scan-action@v5  
      with:  
        args: -Dsonar.projectBaseDir=backend  
      env:  
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}  
        SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}  
    - uses: SonarSource/sonarqube-quality-gate-action@v1  
      with:  
        scanMetadataReportFile: backend/.scannerwork/report-task.txt  
        timeout-minutes: 5  
      env:  
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}  
  
sonarqube_scan_frontend:  
  runs-on: ubuntu-latest  
  steps:  
    - uses: actions/checkout@v4  
    - uses: SonarSource/sonarqube-scan-action@v5  
      with:  
        args: -Dsonar.projectBaseDir=frontend  
      env:  
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}  
        SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}  
    - uses: SonarSource/sonarqube-quality-gate-action@v1  
      with:  
        scanMetadataReportFile: frontend/.scannerwork/report-task.txt  
        timeout-minutes: 5  
      env:  
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```

The test stage follows the linting stage by defining order via the needs keyword, where the jobs wait on all jobs of the previous stage. Both jobs checkout the code base, setup node, install packages and run linting and the jest tests.

```
test-backend:
  name: Backend Unit Tests
  runs-on: ubuntu-latest
  needs: [ sonarqube_scan_backend, sonarqube_scan_frontend, vulnerability_scan ]
  defaults:
    run:
      working-directory: backend
  steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-node@v4
    - run: npm ci
    - run: npm run eslint
    - run: npm run jest
    - name: Archive code coverage results
      uses: actions/upload-artifact@v4
      with:
        name: backend-coverage-report
        path: backend/coverage/lcov-report

test-frontend:
  name: Frontend Unit Tests
  runs-on: ubuntu-latest
  needs: [ sonarqube_scan_backend, sonarqube_scan_frontend, vulnerability_scan ]
  defaults:
    run:
      working-directory: frontend
  steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-node@v4
    - run: npm ci
    - run: npm run eslint
    - run: npm run jest
    - name: Archive code coverage results
      uses: actions/upload-artifact@v4
      with:
        name: frontend-coverage-report
        path: frontend/coverage/lcov-report
```

The next stage is the building and delivering stage. Here, a separate workflow is being called to avoid repeating large sections of steps (between backend and frontend). The definition of “secrets: inherit” is important for the call to access GitHub actions secrets in the called workflow. Variables are passed via “with”.

```
build-deliver-backend-image:
  name: Build & Deliver Backend Image
  needs: test-backend
  secrets: inherit
  uses: ./github/workflows/build-and-deliver.yml
  with:
    working-directory: backend
    dockerhub-repository: secret-notes-backend
    image-artifact-name: backend-image

build-deliver-frontend-image:
  name: Build & Deliver Frontend Image
  needs: test-frontend
  secrets: inherit
  uses: ./github/workflows/build-and-deliver.yml
  with:
    working-directory: frontend
    dockerhub-repository: secret-notes-frontend
    image-artifact-name: frontend-image
```

The workflow file “build-and-deliver.yml” starts with definitions of variables that can be passed.

```
name: Build and Push Docker Image

on:
  workflow_call:
    inputs:
      working-directory:
        required: true
        type: string
      dockerhub-repository:
        required: true
        type: string
      image-artifact-name:
        required: true
        type: string

env:
  DOCKERHUB_USERNAME: nick7152
```

The jobs build an image using the Git commit SHA as the image tag. The image gets pushed to Docker Hub. Additionally, the image is also pushed under the tag “latest”. The username for DockerHub is simply defined as an environment variable here, as it is not secret. The built image gets passed from the build to the deliver job via an artifact.

```
build-image:
  name: Build Docker Image
  runs-on: ubuntu-latest
  outputs:
    image-tag: ${{ steps.tag.outputs.image-tag }}
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Set Docker image tag
      id: tag
      run: echo "image-tag=${GITHUB_SHA}" >> "$GITHUB_OUTPUT"
    - name: Build Docker image
      run: |
        docker build -t ${env.DOCKERHUB_USERNAME}/${inputs.dockerhub-repository}:${steps.tag.outputs.image-tag} ${inputs.working-directory}
    - name: Save Docker image to file
      run: |
        docker save ${env.DOCKERHUB_USERNAME}/${inputs.dockerhub-repository}:${steps.tag.outputs.image-tag} -o image.tar
    - name: Upload image artifact
      uses: actions/upload-artifact@v4
      with:
        name: ${inputs.image-artifact-name}
        path: image.tar
```

```
push-backend-image:
  name: Push Docker Image
  needs: build-image
  runs-on: ubuntu-latest
  steps:
    - name: Download image artifact
      uses: actions/download-artifact@v4
      with:
        name: ${inputs.image-artifact-name}
        path: .
    - name: Load Docker image
      run: |
        docker load -i image.tar
    - name: Log in to Docker Hub
      uses: docker/login-action@v3
      with:
        username: ${env.DOCKERHUB_USERNAME}
        password: ${secrets.DOCKERHUB_TOKEN}
    - name: Push Docker image
      run: |
        docker tag ${env.DOCKERHUB_USERNAME}/${inputs.dockerhub-repository}:${needs.build-image.outputs.image-tag} \
          ${env.DOCKERHUB_USERNAME}/${inputs.dockerhub-repository}:latest
        docker push ${env.DOCKERHUB_USERNAME}/${inputs.dockerhub-repository}:${needs.build-image.outputs.image-tag}
        docker push ${env.DOCKERHUB_USERNAME}/${inputs.dockerhub-repository}:latest
```

The next stage is “deploy-green”. Here a job connects to the docker host EC2 instance via SSH (using an marketplace action). Secrets, such as the username or SSH private key, are stored in the repository’s action’s secrets. On the host, the .env file is updated with the Git commit SHA of the newly built images followed by the Docker deployment of the green branch.

```
deploy-green:
  name: Deploy new Green version
  runs-on: ubuntu-latest
  needs: [ build-deliver-backend-image, build-deliver-frontend-image, update-local-repo ]
  if: github.ref == 'refs/heads/production'
  outputs:
    image-tag: ${{ steps.tag.outputs.image-tag }}
  steps:
    - name: Get commit SHA
      id: tag
      run: echo "image-tag=${GITHUB_SHA}" >> "$GITHUB_OUTPUT"
    - name: Deploy new green version
      uses: appleboy/ssh-action@v1
      env:
        GREEN_BACKEND_TAG: ${{ steps.tag.outputs.image-tag }}
        GREEN_FRONTEND_TAG: ${{ steps.tag.outputs.image-tag }}
      with:
        host: ${{ secrets.DEPLOY_HOST }}
        username: ${{ secrets.DEPLOY_HOST_USERNAME }}
        key: ${{ secrets.DEPLOY_HOST_SSH_KEY }}
        envs: GREEN_BACKEND_TAG, GREEN_FRONTEND_TAG
      script: |
        cd /opt/CONINT && \
        echo 'Deploying Frontend to green...' && \
        docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml down frontend-green && \
        IMAGE_TAG=${GREEN_FRONTEND_TAG} docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml up -d --build frontend-green && \
        echo 'Deploying Backend to green...' && \
        docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml down backend-green && \
        IMAGE_TAG=${GREEN_BACKEND_TAG} docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml up -d --build backend-green
```

Now that the green environment is re-deployed, the E2E test can be conducted.

```
e2e-and-performance-tests:
  name: E2E & Performance Tests
  runs-on: ubuntu-latest
  needs: deploy-green
  defaults:
    run:
      working-directory: frontend
  steps:
    - uses: actions/checkout@v4

    - name: Set up Node.js
      uses: actions/setup-node@v4
      with:
        node-version: 18

    - name: Install Dependencies
      run: npm ci

    - name: Install Playwright Browsers
      run: npx playwright install --with-deps

    - name: Run Playwright E2E Tests
      run: npm run playwright
      env:
        FRONTEND_GREEN: https://staging.conint-securenotes.online

    - name: Install k6
      run: |
        sudo gpg -k
        sudo gpg --no-default-keyring --keyring /usr/share/keyrings/k6-archive-keyring.gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys C5AD17C747E3415A3642057D77C6C491D6AC1D69
        echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list > /dev/null
        sudo apt update
        sudo apt install -y k6

    - name: Run k6 Performance Tests
      env:
        BACKEND_GREEN: https://staging.conint-securenotes.online/api
      run: BACKEND_GREEN=${BACKEND_GREEN} npm run k6
```

In the E2E and performance test stage, the tests are executed against the green environment to validate its functionality before switching traffic to it. It starts with setting up Node.js version 18 as the runtime environment. It installs all frontend dependencies using npm ci and then installs all necessary browsers for Playwright. The Playwright E2E tests are executed with the Base URL set to the staging Url. After the E2E tests, the performance testing tool k6 is installed and the tests are executed.

Only if this stage succeeds, does the pipeline continue with the blue-green switch. This is again accomplished by using an SSH action where the NGINX configs get changed and executing a NGINX reload.

```
switch-green-blue:
  name: Switch from Blue to Green
  runs-on: ubuntu-latest
  needs: e2e-and-performance-tests
  steps:
    - name: Switch from Blue to Green
      uses: appleboy/ssh-action@v1
      with:
        host: ${{ secrets.DEPLOY_HOST }}
        username: ${{ secrets.DEPLOY_HOST_USERNAME }}
        key: ${{ secrets.DEPLOY_HOST_SSH_KEY }}
        script: |
          docker exec blue-green-proxy sh -c "
            sed -i \
              -e 's|set \\$active_frontend http://frontend-blue:80;|set \\$active_frontend http://frontend-green:80;|' \
              -e 's|set \\$active_backend http://backend-blue:3000;|set \\$active_backend http://backend-green:3000;|' \
              /etc/nginx/conf.d/blue-green.conf && \
            nginx -s reload"
```

After this was successful, the blue environment is updated silently to the new images. A switch back to blue is made at last.

```
update-blue:
  name: Update Blue version and switch back to Blue
  runs-on: ubuntu-latest
  needs: switch-green-blue
  outputs:
    image-tag: ${{ steps.tag.outputs.image-tag }}
  steps:
    - name: Get commit SHA
      id: tag
      run: echo "image-tag=${GITHUB_SHA}" >> "$GITHUB_OUTPUT"
    - name: Update Blue version
      uses: appleboy/ssh-action@v1
      env:
        BLUE_BACKEND_TAG: ${{ steps.tag.outputs.image-tag }}
        BLUE_FRONTEND_TAG: ${{ steps.tag.outputs.image-tag }}
    with:
      host: ${secrets.DEPLOY_HOST}
      username: ${secrets.DEPLOY_HOST_USERNAME}
      key: ${secrets.DEPLOY_HOST_SSH_KEY}
      envs: BLUE_BACKEND_TAG,BLUE_FRONTEND_TAG
      script: |
        cd /opt/CONINT && \
        echo 'Deploying Frontend to blue...' && \
        docker-compose -f stacks/secret_notes/docker-compose-secret-notes-blue.yml down frontend-blue && \
        IMAGE_TAG=${BLUE_FRONTEND_TAG} docker-compose -f stacks/secret_notes/docker-compose-secret-notes-blue.yml up -d --build frontend-blue && \
        echo 'Deploying Backend to blue...' && \
        docker-compose -f stacks/secret_notes/docker-compose-secret-notes-blue.yml down backend-blue && \
        IMAGE_TAG=${BLUE_BACKEND_TAG} docker-compose -f stacks/secret_notes/docker-compose-secret-notes-blue.yml up -d --build backend-blue

  - name: Switch back to Blue
    uses: appleboy/ssh-action@v1
    with:
      host: ${secrets.DEPLOY_HOST}
      username: ${secrets.DEPLOY_HOST_USERNAME}
      key: ${secrets.DEPLOY_HOST_SSH_KEY}
      script: |
        docker exec blue-green-proxy sh -c "
          sed -i \
            -e 's|set \\$active_frontend http://frontend-green:80;|set \\$active_frontend http://frontend-blue:80;|' \
            -e 's|set \\$active_backend http://backend-green:3000;|set \\$active_backend http://backend-blue:3000;|' \
            /etc/nginx/conf.d/blue-green.conf && \
          nginx -s reload"
```

If the GitHub actions pipeline fails, the triggering user (usually the issuer of a commit if not triggered manually) gets notified via email automatically.

9.2 Jenkins

While the GitHub Actions Pipeline is triggered through pushes to main or production, Jenkins must be triggered manual. Otherwise, they would conflict with each other. However, Jenkins provides a brighter variety of input variables for our run and a better control of which stages are executed.

A huge benefit compared to GitHub Actions is that the workspace and build results are persistent as well as the Jenkins container was built with predefined dependencies in the Dockerfile. This leads to a faster run since not everything has to be installed every run. Most of the stages are parallel, meaning that Backend and Frontend tasks are executed at the same time to increase efficiency.

Because Jenkins runs as a container on the host and not on the internet, there are some changes in the pipeline configuration like different base URLs for Frontend and Backend or test environments. We use inputs through UI and fixed environment variables. We also define on which agents/nodes we run the pipeline (in our case they are run in the Jenkins container itself):

```
pipeline {
    agent any

    parameters {
        booleanParam(name: 'STATIC_TESTS', defaultValue: true, description: 'Statische Code Analyse durchführen? (lint, snyk, sonar, jest)')
        booleanParam(name: 'DYNAMIC_TESTS', defaultValue: true, description: 'Dynamische Tests durchführen? (playwright, k6)')
        booleanParam(name: 'BUILD_FRONTEND', defaultValue: false, description: 'Frontend bauen und pushen?')
        booleanParam(name: 'BUILD_BACKEND', defaultValue: false, description: 'Backend bauen und pushen?')
        booleanParam(name: 'DEPLOY', defaultValue: false, description: 'Erzeugten Build deployen und testen?')
        string(name: 'IMAGE_TAG', defaultValue: 'latest', description: 'Tag unter dem das Docker image gepushed wird')
    }

    environment {
        SONAR_HOST_URL = 'http://sonarqube:9000'
        FRONTEND_GREEN = 'http://frontend-green:80'
        BACKEND_GREEN = 'http://backend-green:3000/api'
        FRONTEND_IMAGE = 'nick7152/secret-notes-frontend'
        BACKEND_IMAGE = 'nick7152/secret-notes-backend'
    }
}
```

The variables are already explained in 5.2 and won't be explained again here. Quick recap: depending on the variables stages are either executed or skipped. This can be seen through the initial "when" condition in every stage, e.g.:

```
stage('Unit Tests') {
    when {
        expression { return params.STATIC_TESTS }
    }
}
```

The first stage ensures that all npm dependencies are installed (compared to npm install, npm ci always reinstalls which is ideal for CICD tools like Jenkins since it ensures to handle with the same condition in every run. Because the workspace is consistent, node_modules is also consistent):

```
stage('Install Dependencies') {
    parallel {
        stage('Install Frontend Dependencies') {
            steps {
                dir('frontend') {
                    echo 'Installing frontend dependencies...'
                    sh 'npm ci'
                }
            }
        }
        stage('Install Backend Dependencies') {
            steps {
                dir('backend') {
                    echo 'Installing backend dependencies...'
                    sh 'npm ci'
                }
            }
        }
    }
}
```

Now the Lint & Static Analysis can be done (Eslint, SonarQube and Snyk). The credentials which are configured in Jenkins are passed for authentication:

```
stage('Lint & Static Analysis') {
    when {
        expression { return params.STATIC_TESTS }
    }
    parallel {
        stage('Test Frontend') {
            steps {
                dir('frontend') {
                    echo 'Linting Frontend...'
                    sh 'npm run eslint'

                    echo 'SonarQube & Snyk Frontend...'
                    withCredentials([string(credentialsId: 'sonar-creds', variable: 'SONAR_TOKEN')]) {
                        withSonarQubeEnv('SonarQube') {
                            sh 'npm run sonar-scanner'
                        }
                    }
                    timeout(time: 1, unit: 'MINUTES') {
                        waitForQualityGate abortPipeline: true
                    }
                    withCredentials([string(credentialsId: 'snyk-creds', variable: 'SNYK_TOKEN')]) {
                        sh 'npm run snyk-auth'
                    }
                    sh 'npm run snyk'
                }
            }
        }
        stage('Test Backend') {
            steps {
                dir('backend') {
                    echo 'Linting Backend...'
                    sh 'npm run eslint'

                    echo 'SonarQube & Snyk Backend...'
                    withCredentials([string(credentialsId: 'sonar-creds', variable: 'SONAR_TOKEN')]) {
                        withSonarQubeEnv('SonarQube') {
                            sh 'npm run sonar-scanner'
                        }
                    }
                    timeout(time: 1, unit: 'MINUTES') {
                        waitForQualityGate abortPipeline: true
                    }
                    withCredentials([string(credentialsId: 'snyk-creds', variable: 'SNYK_TOKEN')]) {
                        sh 'npm run snyk-auth'
                    }
                    sh 'npm run snyk'
                }
            }
        }
    }
}
```

In a separate stage, the unit tests via Jest are executed:

```
stage('Unit Tests') {
    when {
        expression { return params.STATIC_TESTS }
    }
    parallel {
        stage('Test Frontend') {
            steps {
                dir('frontend') {
                    echo 'Testing Frontend...'
                    sh 'npm run jest'
                }
            }
        }
        stage('Test Backend') {
            steps {
                dir('backend') {
                    echo 'Testing Backend...'
                    sh 'npm run jest'
                }
            }
        }
    }
}
```

If everything succeeded until here the Docker Image is built and tagged with latest and the written tag in the UI through IMAGE_TAG:

```
stage('Build Docker Images') {
    when {
        expression { return params.BUILD_FRONTEND || params.BUILD_BACKEND }
    }
    parallel {
        stage('Build Frontend Image') {
            when {
                expression { return params.BUILD_FRONTEND }
            }
            steps {
                dir('frontend') {
                    echo 'Building Frontend...'
                    sh "docker build -t $FRONTEND_IMAGE:$IMAGE_TAG -t $FRONTEND_IMAGE:latest ."
                }
            }
        }
        stage('Build Backend Image') {
            when {
                expression { return params.BUILD_BACKEND }
            }
            steps {
                dir('backend') {
                    echo 'Building Backend...'
                    sh "docker build -t $BACKEND_IMAGE:$IMAGE_TAG -t $BACKEND_IMAGE:latest ."
                }
            }
        }
    }
}
```

With the configured credentials they are sent to Docker Hub:

```
stage('Push Docker Images') {
    when {
        expression { return params.BUILD_FRONTEND || params.BUILD_BACKEND }
    }
    parallel {
        stage('Push Frontend Docker Image') {
            when {
                expression { return params.BUILD_FRONTEND }
            }
            steps {
                withCredentials([usernamePassword(credentialsId: 'dockerhub-creds', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                    sh "echo $DOCKER_PASS | docker login -u $DOCKER_USER --password-stdin"
                    echo 'Pushing Frontend...'
                    sh "docker push $FRONTEND_IMAGE:latest"
                    sh "docker push $FRONTEND_IMAGE:$IMAGE_TAG"
                }
            }
        }
        stage('Push Backend Docker Image') {
            when {
                expression { return params.BUILD_BACKEND }
            }
            steps {
                withCredentials([usernamePassword(credentialsId: 'dockerhub-creds', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                    sh "echo $DOCKER_PASS | docker login -u $DOCKER_USER --password-stdin"
                    echo 'Pushing Backend...'
                    sh "docker push $BACKEND_IMAGE:latest"
                    sh "docker push $BACKEND_IMAGE:$IMAGE_TAG"
                }
            }
        }
    }
}
```

Since the following tasks are only responsible for blue/green deployment for Frontend and Backend, a quick stage was included to ensure a running database for the first deploy:

```
stage('Start Database if not running') {
    when {
        expression { return params.DEPLOY }
    }
    steps {
        echo 'Starting database...'
        sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml up -d db"
    }
}
```

The method of blue/green deployment is the same as in GitHub Actions, a local docker compose green file which comes from the repository is executed. Here the IMAGE_TAG is passed once again and is included in the docker compose green file:

```
stage('Deploy to Green') {
    when {
        expression { return params.DEPLOY }
    }
    parallel {
        stage('Deploy Frontend to Green') {
            steps {
                echo 'Deploying Frontend to green...'
                sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml down frontend-green"
                sh "IMAGE_TAG=${IMAGE_TAG} docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml up -d --build frontend-green"
            }
        }
        stage('Deploy Backend to Green') {
            steps {
                echo 'Deploying Backend to green...'
                sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml down backend-green"
                sh "IMAGE_TAG=${IMAGE_TAG} docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml up -d --build backend-green"
            }
        }
    }
}
```

If the deployment on green was successful, we can start with final testing. As mentioned, Jenkins was already built with most of the dependencies, therefore we do not need to install playwright or k6. Because playwright accesses the Frontend and k6 the Backend, environment variables need to be passed. For playwright it's enough to declare the URL in the beginning of the Jenkinsfile as an environment variable, but for k6 we need to pass it on:

```
stage('E2E & Performance Testing') {
    when {
        expression { return params.DYNAMIC_TESTS }
    }
    steps {
        dir('frontend') {
            echo 'Testing Frontend on green...'
            sh 'npm run playwright'
            sh "BACKEND_GREEN=${BACKEND_GREEN} npm run k6"
        }
    }
}
```

The last stages are the same as in GitHub Actions: switch to green, deploy to blue, switch to blue:

```
stage('Switch to Green') {
    when {
        expression { return params.DEPLOY }
    }
    steps {
        echo 'Switching from blue to green deployment...'
        sh ''
        docker exec blue-green-proxy sh -c "
            sed -i \
                -e 's|set \\$active_frontend http://frontend-blue:80;|set \\$active_frontend http://frontend-green:80;|' \
                -e 's|set \\$active_backend http://backend-blue:3000;|set \\$active_backend http://backend-green:3000;|' \
                /etc/nginx/conf.d/blue-green.conf &&
            nginx -s reload
        "
    }
}
```

```

stage('Deploy to Blue') {
    when {
        expression { return params.DEPLOY }
    }
    parallel {
        stage('Deploy Frontend to Blue') {
            steps {
                echo 'Deploying Frontend to blue...'
                sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-blue.yml down frontend-blue"
                sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-blue.yml up -d --build frontend-blue"
            }
        }
        stage('Deploy Backend to Blue') {
            steps {
                echo 'Deploying Backend to blue...'
                sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-blue.yml down backend-blue"
                sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-blue.yml up -d --build backend-blue"
            }
        }
    }
}

stage('Switch to Blue') {
    when {
        expression { return params.DEPLOY }
    }
    steps {
        echo 'Switching green to blue deployment...'
        sh ''
        docker exec blue-green-proxy sh -c "
            sed -i \
                -e 's|set \$active_frontend http://frontend-green:80;|set \$active_frontend http://frontend-blue:80;|' \
                -e 's|set \$active_backend http://backend-green:3000;|set \$active_backend http://backend-blue:3000;|' \
                /etc/nginx/conf.d/blue-green.conf && \
            nginx -s reload
        "
        ...
    }
}

```

The result of the run can be seen in Jenkins either on the pipeline dashboard with the Pipeline: Stage View Plugin or in the Blue Ocean view, where also parallel stages are shown correctly.

10. Features and Refined User Stories

- Detailed acceptance criteria
- Implementation details

Detailed acceptance criteria and implementation details

User story 1:

The user enters text and a key/passphrase

Notes App

Alle Notizen

Neue Notiz erstellen

Titel

Inhalt

Verschlüsselungs-Key

The note is encrypted on the server and stored in the database

```
export function encryptNote(secretKey, plaintext) { Show usages ▾ Rene Rumetshofer
  const salt = crypto.randomBytes(16);
  const key = deriveKey(secretKey, salt);

  const iv = crypto.randomBytes(12);
  const cipher = crypto.createCipheriv('aes-256-gcm', key, iv);

  const encrypted = Buffer.concat([
    cipher.update(plaintext, 'utf8'),
    cipher.final(),
  ]);
  const authTag = cipher.getAuthTag();

  // Data = salt + iv + authTag + cipher text
  return Buffer.concat([salt, iv, authTag, encrypted]).toString('base64');
}
```

```
fastify.post('/api/notes', postNoteOpts, async (request, reply) :Promise<any> => {
  const encryptedContent = encryptNote(
    request.body.key,
    request.body.content
  );
  const noteUuid = generateUUID();
  return fastify.pg.transact(callback: async (client) :Promise<void> => {
    await client.query(
      'INSERT INTO notes(notes_uuid, title, content) VALUES($1, $2, $3)',
      [noteUuid, request.body.title, encryptedContent]
    );
    reply.code(200).send({ notes_uuid: noteUuid });
  });
});
```

UI confirms that the note is created securely



User Story 2:

The user enters the same key used to encrypt

Titel	UUID: af398624-56a7-47e3-8c11-6f69a0eeab3a
123	
Anzeigen	Löschen
Test	

The system decrypts the note if the key is valid, returning the plaintext to the user

```
export function decryptNote(secretKey, encryptedData) { Show usages
  const data = Buffer.from(encryptedData, 'base64');

  const salt = data.subarray(0, 16);
  const iv = data.subarray(16, 28);
  const authTag = data.subarray(28, 44);
  const ciphertext = data.subarray(44);

  const key = deriveKey(secretKey, salt);
  const decipher = crypto.createDecipheriv('aes-256-gcm', key, iv);
  decipher.setAuthTag(authTag);

  const decrypted = Buffer.concat([
    decipher.update(ciphertext),
    decipher.final(),
  ]);
  return decrypted.toString('utf8');
}
```

The system rejects or returns an error if the key is invalid

Titel	UUID: af398624-56a7-47e3-8c11-6f69a0eeab3a
342	
Anzeigen	Löschen
Fehler: 403 Key is invalid: Error: Unsupported state or unable to authenticate data	

User Story 3:

A toggle in PostHog allows enabling or disabling a second UI

```
if (typeof posthog !== "undefined") {  
  /* eslint-disable no-undef */  
  posthog.onFeatureFlags(() : void => {  
    const variant = posthog.getFeatureFlag("new-ui-theme");  
    /* eslint-enable no-undef */  
    const button : HTMLElement = document.getElementById(elementId: "createNote");  
    if (button) {  
      button.style.backgroundColor = variant === "variant" ? "green" : "blue";  
    }  
  });  
}
```

A/B testing splits users into at least two groups with distinct experiences

As seen in the screenshot above, users are split into two groups and their UI is different, although just the button colour.

11. A/B Testing

- How to set up a new split test for a different UI feature or encryption flow

The feature flag is of type “Multiple variants with rollout percentages” and contains the variant keys “control” and “variant”, both of which have an equal rollout of 50%, randomly determined. “Control” version has a blue button and “variant” has a green button.

Here, the respective version is determined.

```
if (typeof posthog !== "undefined") {
  posthog.onFeatureFlags(() => {
    const variant = posthog.getFeatureFlag("new-ui-theme");

    const button = document.getElementById("createNote");
    if (button) {
      button.style.backgroundColor = variant === "variant" ? "green" : "blue";
    }

    window._variant = variant;
  });
}
```

Example for “control”

Notes App

Alle Notizen

Neue Notiz erstellen

Titel

Inhalt

Verschlüsselungs-Key

Erstellen

Example for “variant”

Notes App

Alle Notizen

Neue Notiz erstellen

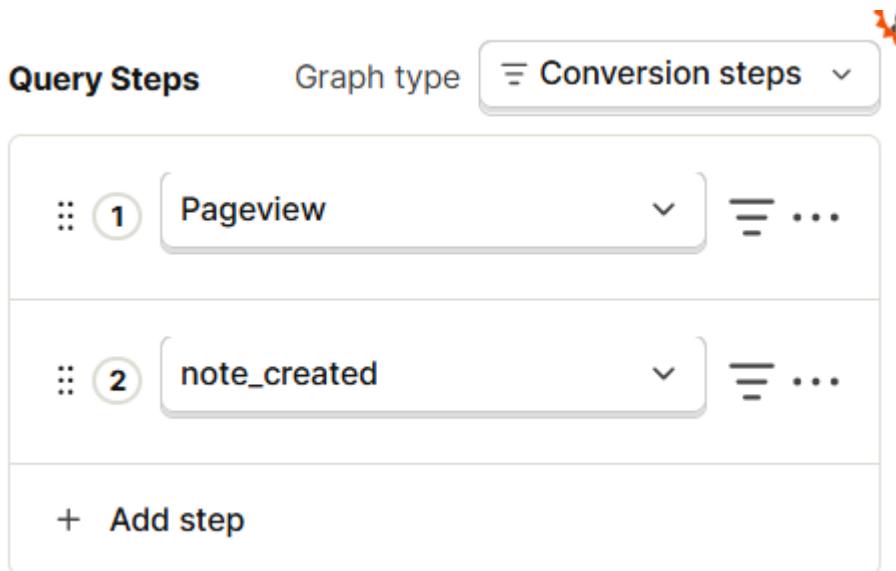
Titel

Inhalt

Verschlüsselungs-Key

Erstellen

With the help of funnels created in PostHog, we decided that a good user conversion rate is determined by how many users create a note upon entering the website. The funnel's query steps were chosen to be Pageview --> “note_created”.



“note_created” is an event that was created to track when a user has clicked on “Erstellen”. In it, an event property that is used as a filter to distinguish between both user groups is created.

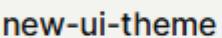
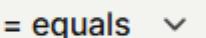
```
if (typeof posthog !== "undefined") {
  posthog.capture("note_created", {
    "new-ui-theme": window._variant || "unknown"
  });
}
```

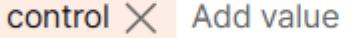
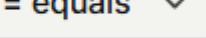
In filters, you can further specify which funnel of the versions you want to investigate.

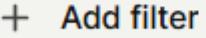
Filters

Filter out internal and test users  

Match  all  filters in this group  





However, due to complications, PostHog was not able to read the event property "new-ui-theme" that is tracked by the capture function. This is why the funnel below is showing each query step, with all users combined, instead of each user group in comparison to each other.



12. Blue/Green Deployment

- How you implement it on AWS (e.g., Docker Compose definitions swapped behind a load balancer)

Blue/Green deployment is implemented in a manner that only the Docker host is used for hosting. The following routes have been defined:

- “staging.conint-securenotes.online” is always routed to the green environment
- “prod.conint-securenotes.online” is routed to blue or green, depending on the current state of a deployment.

Incoming traffic is routed as follows: The first responding service is NGINX Proxy Manager which handles TLS certification. Traffic is forwarded to another NGINX instance called “blue-green-proxy” which handles routing between blue and green, and in the case of the staging route to green:

```
blue-green-proxy:
  image: nginx:1.29-alpine
  container_name: blue-green-proxy
  restart: unless-stopped
  expose:
    - 80
  volumes:
    - /opt/CONINT/stacks/reverse_proxy/nginx/conf.d:/etc/nginx/conf.d/
    - /opt/CONINT/stacks/reverse_proxy/nginx/nginx.conf:/etc/nginx/nginx.conf
  networks:
    - secret_notes_secret_notes_network
    - proxy_network
```

Routing for “prod”

```
server {
  listen 80;
  server_name prod.conint-securenotes.online;

  set $active_frontend http://frontend-blue:80;
  set $active_backend http://backend-blue:3000;

  location / {
    proxy_pass $active_frontend;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
  }

  location /api {
    proxy_pass $active_backend;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
  }
}
```

Routing for “staging”:

```
server {
    listen 80;
    server_name staging.conint-securenotes.online;

    set $frontend http://frontend-green:80;
    set $backend http://backend-green:3000;

    location / {
        proxy_pass $frontend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api {
        proxy_pass $backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

To modify the routing for prod to point to either green or blue, the file is modified during the CICD run:

```
docker exec blue-green-proxy sh -c "
sed -i \
-e 's|set \\$active_frontend http://frontend-blue:80;|set \\$active_frontend http://frontend-green:80;|' \
-e 's|set \\$active_backend http://backend-blue:3000;|set \\$active_backend http://backend-green:3000;|' \
/etc/nginx/conf.d/blue-green.conf && \
nginx -s reload"
```

The stack of containers needed for running blue/green frontend and backend as well as the proxy is managed in a docker compose file on the server:

```
backend-blue:
  image: nick7152/secret-notes-backend:${IMAGE_TAG}
  container_name: backend-blue
  restart: unless-stopped
  env_file:
    - docker.env
  expose:
    - 3000
  networks:
    - secret_notes_network
frontend-blue:
  image: nick7152/secret-notes-frontend:${IMAGE_TAG}
  container_name: frontend-blue
  restart: unless-stopped
  expose:
    - 80
  networks:
    - secret_notes_network
```

```
backend-green:
  image: nick7152/secret-notes-backend:${IMAGE_TAG}
  container_name: backend-green
  restart: unless-stopped
  env_file:
    - docker.env
  expose:
    - 3000
  networks:
    - secret_notes_network
    - jenkins_jenkins_network

frontend-green:
  image: nick7152/secret-notes-frontend:${IMAGE_TAG}
  container_name: frontend-green
  restart: unless-stopped
  expose:
    - 80
  networks:
    - secret_notes_network
    - jenkins_jenkins_network
```

All containers, including the NGINX proxy manager, are part of an external network “proxy_network” for enabling communication between them.

When new tags of images are available, they will replace the old tag during the rollout process. This is done via environment variables which are passed on through GitHub Actions or Jenkins directly in the shell. Either the tag was entered in the UI (Jenkins) or auto generated (GitHub Actions)

```
parallel {
  stage('Deploy Frontend to Green') {
    steps {
      echo 'Deploying Frontend to green...'
      sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml down frontend-green"
      sh "IMAGE_TAG=${IMAGE_TAG} docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml up -d --build frontend-green"
    }
  }
  stage('Deploy Backend to Green') {
    steps {
      echo 'Deploying Backend to green...'
      sh "docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml down backend-green"
      sh "IMAGE_TAG=${IMAGE_TAG} docker-compose -f stacks/secret_notes/docker-compose-secret-notes-green.yml up -d --build backend-green"
    }
  }
}
```

Two proxy hosts have been set up on the NGINX proxy manager instance:

 staging.conint-securenotes.online Created: 21st June 2025	http://blue-green-proxy.deployment-network:81	Let's Encrypt	Public	● Online	⋮
 prod.conint-securenotes.online Created: 21st June 2025	http://blue-green-proxy.deployment-network:80	Let's Encrypt	Public	● Online	⋮

The mapping to port 81 for the staging route can be seen here.

13. Logging and Monitoring (Outlook)

- Proposed approach for logging, analytics, and potential monitoring solutions (e.g., CloudWatch, Splunk, ELK)

Logging

For centralized logging, we would integrate the ELK Stack to collect and visualize application logs from both frontend and backend. This would allow us to track errors and debug issues more efficiently.

Analytics

For user analytics we would use PostHog on the frontend to track the user behaviour, feature usage and conversion metrics. Backend events such as note creation or authentication can be logged in a structured format for analysis.

Monitoring

To monitor system performance and availability, we would use CloudWatch for system-level and custom application metrics.