1.0.3 Settings for networking and connectivity

Автор: Oleg Kravchuk
Последнее обновление: мар. 22, 2022

Networking

Для общения между командами используется корпоративный мессенджер Slack. Для постановки задач и отслеживания прогресса проекта у команды разработки используется Jira, а у команды дизайна используется Asana. Для просмотра расписания встреч команд используется Google calendar, который привязывается к корпоративной электронной почте (briolink.com). Для хранения исходного кода проекта используется GitLab. Figma — онлайн-сервис для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени.

Slack

Slack — платформа для корпоративного общения и работы в команде над проектами. Цель сервиса — оптимизировать коммуникацию в команде и облегчить взаимодействие между отделами. Благодаря интеграции слака со множеством программ и сервисов он позволяет компании создать полноценную рабочую среду.

Компании в Slack создают для себя рабочее пространство, в котором содержатся каналы, архивы

документов и ленты новостей. Каналы — групповой чат, где ведётся общение отдельной команды и решаются вопросы по заданной теме. В чатах пользователи могут общаться при помощи текстовых сообщений, используя специальные средства для форматирования текста: выделение *курсивом*, **полужирным** или подчёркиванием, создание списков, оформление программного кода и другое. Помимо текстового общения в Slack есть голосовые и видео звонки. Партнёров и сторонних людей можно приглашать в отдельные каналы. Пользователи могут общаться не только в групповых чатах, но и личных.

Для подключения в рабочее пространство требуется приглашение от @Andrey Kunov . После приглашения Вам нужно подключиться в следующие каналы:

Общие каналы:

General — канал, в котором анонсируются объявления и обсуждаются вопросы, касающиеся всей команды.

Testing — канал предназначенный для обсуждения, выявленных багов при тестировании продукта.

Daily-reports — канал для ежедневных отчетов о выполненных задачах.

Каналы для разработчиков:

Dev — канал для обсуждения вопросов разработки, текущего статуса, объявлений для команды разработки.

Design-and-dev — канал для обсуждения вопросов между дизайнерами и командой разработки.

Каналы для дизайнеров:

Design — канал для обсуждение дизайн-тем, касаемых разработки продукта Briolink.

Network-research — канал для обсуждения статей для размещения на сайт Briolink.

Website — канал для обсуждения наполнения контентом сайта Briolink.

Whiteboard — канал обсуждения дальнейшего развития продукта.

Design-and-dev — общий канал с разработчиками, где обсуждается решения на счет MVP продукта.

Личные переписки:

В данной таблице описываются роли в проекте и по каким вопросам можно обратиться:

Имя пользователя	Роль	Вопросы
Andrey Kunov	Founder & President	Связанные с работой, рабочим графиком, контрактом.
Dmitriy Mun	Product manager, Designer	Связанные по бизнес-логике продукта, по дизайну продукта.
Sergey Zaburunov	Director of Engineering	Связанные с разработкой, технические вопросы.

Olena Bondareva	Finance Director	Связанные с оплатой по контракту, заключением договоров.

Jira

Jira — это инструмент для планирования спринтов, создания и распределения задач, отлеживания ошибок, а также ведения agile-проектов. Сервис доступен как в веб-версии, так и в виде десктопного приложения.

Чтобы присоединиться к проекту требуется приглашение от @Andrey Kunov .

Проект называется “BL Network MVP (BNM)”.

В данном разделе описаны основные моменты с работой в Jira.

Основные понятия:

Sprint — это короткий временной интервал, в течение которого команда выполняет запланированный объем работы. Продолжительность sprint составляет 1 рабочая неделя (с понедельника по понедельник).

Backlog — это упорядоченный и постоянно обновляемый список всего, что планируется сделать для создания и улучшения продукта.

Roadmap — это стратегия долгосрочного развития продукта или решения. С помощью дорожных

карт владельцы продуктов описывают будущие функциональные возможности продукта и устанавливают сроки, к которым эти новые возможности будут выпущены.

Epic — это большая часть работы, которую можно разбить на несколько небольших задач. Часто они включают в себя несколько спринтов.

Task — обозначает любое конечное задание, которое можно описать коротким императивом. В данном проекте task используется как задача, результат которой не виден для конечного пользователя. Обычно этот тип задач для backend.

Story — задача, результат которой виден пользователю. Обычно этот тип назначается для frontend.

Названия задач

1. Тип задачи — story, task, bug.
2. К чему относиться задача — FE (FrontEnd), BE (BackEnd).
3. Название сервиса — Connection, User, Company и т.д.
4. Название задачи — New Design “Create Project”, Create connection between users.
5. К какому эпику относиться задача — Connection CRUD, Documentation, User Profile и т.д.

Обычно эти задачи относятся или к микро-сервисам, или микро-фронтедам. Если задача ни к чему не относятся, то она находиться в бэклоге. Время потраченное на daily meeting должно вноситься в эту задачу.

Планирование спринта

Каждый понедельник проходит sprint meeting, где обсуждаются:

Каждый понедельник проходит sprint meeting, где обсуждаются:

1. Результаты и проблемы текущего спринта.
2. Планирование нового спринта с постановкой и распределением задач.

У сотрудника должно набраться 40 часов в неделю, учитывая суммарное время созвонов (обычно около 5-ти часов в неделю).

Для планирования новой задачи нужно:

- Original estimate — время, которое планируется потратить на задачу.
- Учитывать готовность предыдущих задач к имплементации новой (не создан дизайн/функция в другом микро-сервисе).

После планирования задач спринт переноситься во вкладку — “Активные спринты”.

Google Calendar

Это сервис для планирования созвонов, встреч, событий и дел. Чтобы получить к нему доступ нужен аккаунт от корпоративной почты Gmail, который можно получить от @Andrey Kunov . В нём находится актуальная информация о запланированных митингах.

GitLab

GitLab — веб-приложение и система управления репозиториями программного кода для Git.

GitLab предлагает решение для хранения кода и совместной разработки масштабных программных проектов. Репозиторий включает в себя систему контроля версий для размещения различных цепочек разработки и веток, позволяя разработчикам проверять код и откатываться к стабильной версии софта в случае непредвиденных проблем.

версии софта в случае непредвиденных проблем.

Чтобы присоединиться, к проекту Briolink в GitLab, нужно получить приглашение от @Andrey Kunov или @Sergey Zaburunov .

Основная кодовая база находится в подгруппе “network”, в которой находится микро-сервисы и микро-фронтенды.

В каждой подгруппе, есть свой микро-сервис и микро-фронтенды(модули). Чтобы понимать архитекуру и взаимодействия между всеми сервисами, стоит ознакомиться с https://briolink.atlassian.net/wiki/spaces/BD/pages/593985780 .

Feature Branch Workflow

В своей простейшей форме репозиторий мог бы иметь основную ветку со стабильным, доступным кодом и другими ветками для разных фич (или багов, или улучшений), которые можно бы было интегрировать в главную ветку. То есть репозиторий будет иметь второстепенную основную ветку (dev) которая будет хранить тестируемый стабильный код для отправки пользователям, когда он будет слит с master. В этом случае ветка с фичами будет слита с dev, а не с master.

В каждом проекте, есть несколько основных веток:

- dev — ветка для http://dev.briolink.com, development environment используется для тестирования разработчиками новых фич.
- test — ветка для http://test.briolink.com, test environment используется, для всей команды тестирования, когда dev был протестирован разработчками, версия мержится в test.
- demo — ветка для http://demo.briolink.com, demo environment используется для показа функциоанала, у demo находиться реальные данные, который были сделаны для показа, потенциальным клиентам(инвесторам). После тестирования test версия мержится в demo.
- main — ветка для http://app.briolink.com, prod environment, версия для всеобщего пользования продукта, на этотом environment нет изначально никаких данных. Мержиться в prod, обязательно из test и когда была дана команда релиза.

Для того чтобы разработать новую фичу или исправить баг, нужно действовать следующим правилам:

1. Задача должна находиться в Jira, знать её номер.

2. Находясь в dev ветке, создать новую ветку с названием задачи, шаблон названия веток предоставлен ниже:

```
[feature|bugfix]/BNM-[NUMBER]_[NAME TASK]

Example:
feature/BNM-554_new_design_create_project
bugfix/BNN-345_error_upload_image


# Current brauch - dev
git checkout -b feature/BNM-554_new_design_create_project
```

3. Коммиты должны быть оформлены в таком шаблоне

```
BNM-[NUMBER] [Description commit]

Example:
git commit -m "BNM-554 Add modal modal view after create project"
```

4. После выполнения задачи, необходимо создать merge request в dev ветку:

Reviewer оставлять, кто отвественен за проект, в данном примере, описано, что была задача

permission integration в company-service, ответственнен за company @Oleg Kravchuk

5. После принятия merge request — оттестировать новую фичу в dev environment

Common library

Некоторые микро-сервисы используют общие библеотеки, который используется в gradle dependencies. В данном разделе кратко описаны пакеты, как их подключить и как их использовать.

Исходный код пакетов, можно посмотреть в группе Briolink → network → backend. В проекте используется такие библеотеки:

Id project	Name	Description
29889174	Event lib	Данная библеотека служит для отправки сообщений в SNS. С её помощью просходит обещение между микро-сервисами с помощью Domain.
33422039	Location lib	Позволяет запрашивать информацию о местоположении, предоставлять autocomplete для местоположения.
32844103	Permission lib	Позволяет запрашивать информацию о правах пользователя к определенному объекту(Company, CompanyService и т.д.). Добавлять права, удалять их.
33688770	Sync lib	Вносит функционал синхронизации между сервисами.

Подключение в проект

Чтобы подключить пакет в gradle нужно знать id проекта, его можно узнать в gitlab

Узннать актуальную версию пакета, для этого необходимо зайти в проект, в левом меню во вкладку package & registry

Знать deploy-token его можно узнать у разработчиков.

После этого в build.gradle.kts добавить maven репозиторий:

После этого в build.gradle.kts добавить maven репозитории:

- uri — `https://gitlab.com/api/v4/projects/{ID PROJECT}/packages/maven`
- authetication — добавить header:
  Deploy-Token → {DEPLOY_TOKEN}, также не забыть его прокинуть в dockerfile.

```
repositories {
    setOf(
        29889174, // BL Event
        32844103, // BL Permission
    ).forEach {
        maven {
            url = uri("https://gitlab.com/api/v4/projects/$it/packages/maven")
            authentication {
                create<HttpHeaderAuthentication>("header")
            }
            credentials(HttpHeaderCredentials::class) {
                name = "Deploy-Token"
                value = System.getenv("CI_DEPLOY_PASSWORD")
            }
        }
    }
```

После подключения репозитория добавить dependencies

```
dependencies {
    implementation("com.briolink.lib:event:${Versions.BRIOLINK_EVENT}")
    implementation("com.briolink.lib:permission:${Versions.BRIOLINK_PERMISSION}")
}
```

Connectivity

Все пароли и информация, которая будет использоваться в этом разделе, будет доступна в Secrets manager, под названием {NAME_CLUSTER}/bl-network/security-config.

В нём доступны такие параметры:

- **deployToken** — для публикации и загрузки Package Registry gitlab.
- **k8sToken** — токен для входа в kubernetes dashboard.
- **dbUsername** — username для входа в базу данных.
- **dbPassword** — пароль для входа в базу данных.
- **dbHost** — endpoint для подключения к базе данных.
- **hostSshTunnel** — endpoint для подключения к серверу.

bl-general-rsa — private ssh ключ, который можно скачать GitLab repository security

Database

Для подключения к базе данных, используется ssh tunnel:

Parameter	Value
Host	{HOST_SSH_TUNNEL}:22

User name	ec2-user
Authentication type	Key pair
Private key file	{bl-general-rsa}

Параметры для подключения к базе данных:

Parameter	**Value**
Host	{DB_HOST}
Port	5432
User	{DB_USERNAME}
Password	{DB_PASSWORD}

Keycloak

Данные для входа в keycloak хранится в secrets manager под названием {ENVIRONMENT}/bl-network /keycloak-service.

Endpoint для входа: https://{ENVIRONMENT}.briolink.com/

В Secrets доступны значения:

- keycloakUsername — имя пользователя для входа

- keycloakUsername — имя пользователя для входа.
- keycloakPassword — пароль для входа.

1.0.9 Deployment process DEV/TEST/DEMO

Автор: Oleg Kravchuk
Последнее обновление: мар. 14, 2022, Maxim Seshuk

Для деплоя микро-сервиса необходимо следующее:

- [ ] Создать новую запись в Secrets manager
- [ ] Создать базу данных
- [ ] Создать SQS (optional)
- [ ] Настроить helm
- [ ] Настроить werf
- [ ] Настроить dockerfile
- [ ] Настроить giltab-ci

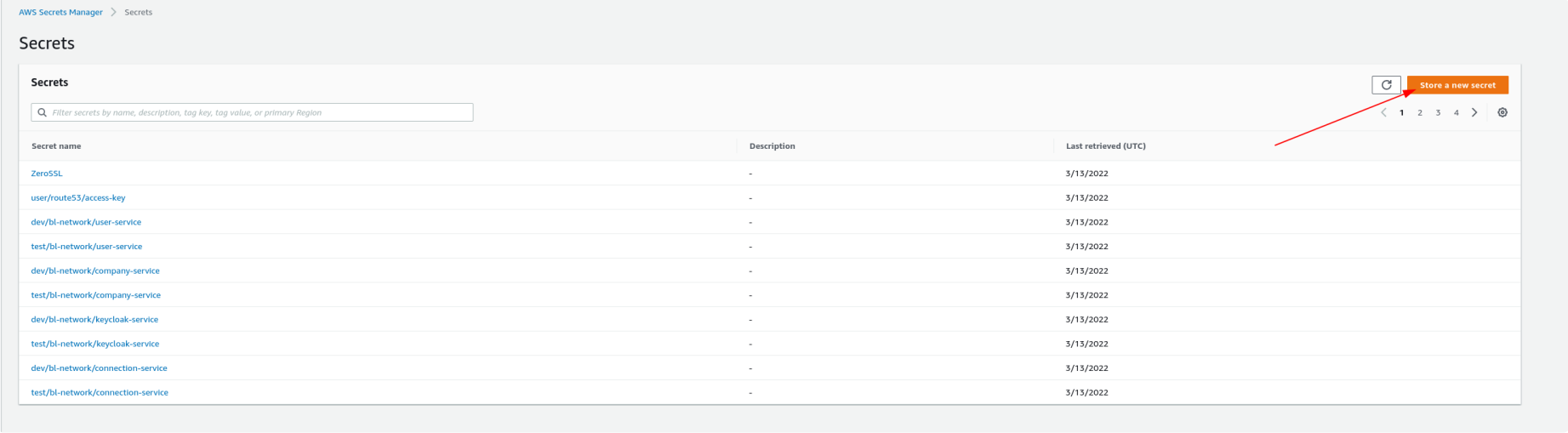
Для деплоя микро-фронтенда необходимо следующее:

- [ ] Настроить werf
- [ ] Настроить helm
- [ ] Настроить dockerfile
- [ ] Настроить giltab-ci

1.0.8.1 Создание записи в secrets manager

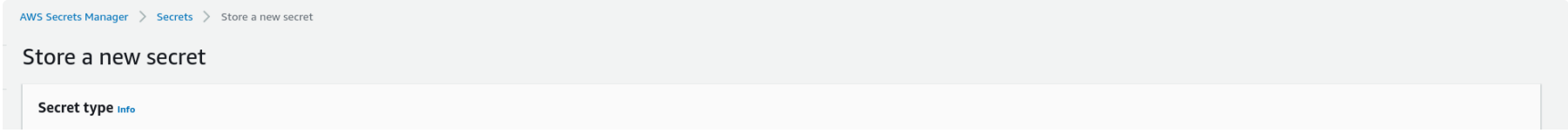
Для создания записи в secrets manager, необходимо:

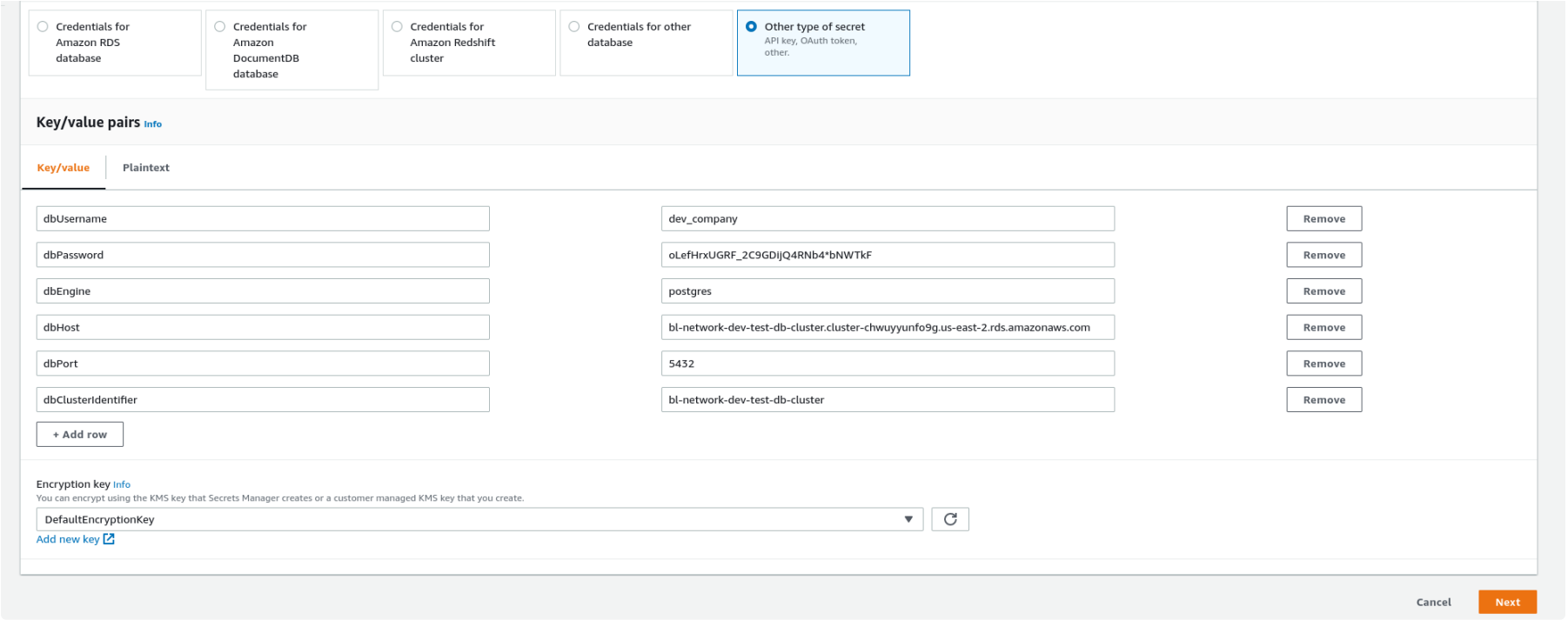
1. На аккаунте, должны быть права для добавления записи secrets manager, для получения прав, обращаться к @Sergey Zaburunov
2. Зайти в AWS Secrets Manager
3. Чтоб запустить работу мастера, нажать на кнопку “Store a new secret”



AWS Secrets manager

4. Step 1 “Choose secret type”:
   a. Secret-type: Other type of secret
   b. Key/value pairs Description key/value, example
   c. Encryption key: DefaultEncryptionKey





Example step 1 “Choose secret type”

5. Step 2 “Configure secret”
    a. Secret name: {ENVIROMENT}/bl-network/{NAME_MICROSERVICE}. Ex: **dev/bl-network/company-service**

Example step 2 “Configure secret”

6. Step 3 “Configure rotation - optional”, здесь необходимо пропустить шаг и приступить к следующему шагу
7. Step 4 “Review”, проверить и нажать кнопку “Store”

Key	**Value example**	**Description**
dbUsername	dev_company; test_connection; demo_user	Название роли / название базы данных Значение составляется из шаблона: {ENVIROMENT}_{NAME_MICROSERVICE}
dbPassword	oLefHrxUGRF_2C9GDijQ4RNb4*bNWTkF	Пароль для базы данных, сгенерировать с 32 символов (рекомендация)
dbEngine	postgres	Название движка для БД
dbHost	bl-network-dev-test-db-	URL для подключения к БД

dbHost	bl-network-dev-test-db-cluster.cluster-chwuyyunfo9g.us-east-2.rds.amazonaws.com	URL для подключения к БД
dbPort	5342	Порт для подключения к БД
dbClusterIdentifier	bl-network-dev-test-db-cluster	Название кластера

1.0.8.2 Создание базы данных

Для создания базы данных необходимо сделать следующее

1. Подключиться к базе данных по dbHost, используя ssh tunnel
2. Создать базу данных, имя базы данных: dbUsername
3. Создать пользователя, имя пользователя: dbUsername и паролем dbPassword
4. Предоставить права " `CONNECT, CREATE, TEMPORARY` " для пользователя к БД
5. Если в сервисе используются схемы, то нужно создать их
6. Добавить расширение `uuid-ossp` `CREATE EXTENSION IF NOT EXISTS "uuid-ossp"`

```
CREATE USER dev_company WITH PASSWORD 'oLefHrxUGRF_2C9GDijQ4RNb4*bNWTkF';
CREATE DATABASE dev_company;
GRANT CONNECT, CREATE, TEMPORARY ON DATABASE dev_company to dev_company;

-- Connect to database dev_company
```

```
6  CREATE SCHEMA read;
7  CREATE SCHEMA write;
8  CREATE EXTENSION IF NOT EXISTS "uuid-ossp"
```

1.0.8.3 Создание очереди SQS

Для добавления очереди необходимо:

1. На аккаунте, должны быть права для создания очереди, для получения прав, обращаться к @Sergey Zaburunov
2. Зайти в AWS Simple Queue Service
3. Чтоб запустить работу мастера, нажать на кнопку “Create queue”
4. Конфигурация
    a. Details
        i. Type: FIFO
        ii. Name: bl-network-{ENVIROMENT}-{NAME_SERVICE}-service.fifo . Ex.: **bl-network-dev-company-service.fifo**

b. Configuraiton

i. Visibility timeout: 30 second

ii. Message retention period: 4 days

iii. Delivery delay: 0 second

iv. Maximum message size: 256 KB

v. Receive message wait time: 0 second

vi. On Content-based deduplication

vii. On “Enable high throughput FIFO”

c. Access policy

i. Method: Advanced

ii. Вставить конфигурацию и заменить company на {NAME_SERVICE}, dev на {ENVIROMENT}

5. Нажать на кнопку “Save”

6. После сохранения очереди, нужно подписать созданную очередь на SNS

a. Нажать на кнопку “Subscribe to Amazon SNS topic”

b. Выбрать topic с текущим {ENVIROMENT}

c. Нажать на кнопку “Save”

- Configuration SQS

1.0.8.4 Настройка helm

В корень микросервиса, необходимо создать папку .helm, добавить файлы:

deployment.yaml

В даном файле описывается конфигурация разворачивания модулей сервиса в kubernetes. Здесь описывается 2 модуля сервиса: Api, Updater

- deployment.yaml microservice
- deployment.yaml microfrontend

Ingress.yaml

В данном файле описывается обеспечение гибкого способа маршрутизации трафика от вашего кластера к внутренним сервисам Kubernetes. Данный файл нужен, чтобы обеспечивать доступ для frontend и API был открыт внешне. Не рекомендуется его добавлять, если микро-сервис пользуется только для внутреннего пользования (location-service, permission-service)

- ingress.yaml

Secrets.yaml

В данном файле описывается конфигурация, для подключения к AWS Secrets manager

- secrets.yaml microservice

Serviceaccount.yaml

В данном файле описывается конфигурация ServiceAccount для авторизации сервиса в kubernetes

- serviceaccount.yaml microservice

service.yaml

В данном файле описывается конфигурация сервиса в kubernetes

- service.yaml microservice

- service.yaml microfrontend

1.0.8.5 Настройка werf

В корне проекта создать 3 файла. CI_DEPLOY_PASSWORD token служит для загрузки внутренних library gitlab package registry.

werf.yaml

В данном файле описывается конфигурация для deploy в kubernetes. Описание атрибутов

- werf.yaml microservice

Werf для фронтенда, изменить название модуля(company) на {NAME_SERVICE}

- werf.yaml microfrontend

werf-giterminism.yaml

Описание атрибутов

- service.yaml

1.0.8.6 Настройка dockerfile

Для каждого модуля(updater, api) микро-сервиса, необходимо создать в корне модуля файл Dockerfile, со следующим содержанием.

- Dockerfile - microservice api
- Dockerfile - microservice updater

Для настройки микро-фронтенда, в корень проекта создать файл

- Dockerfile - micro-frontend

1.0.8.7 Настройка gitlab-ci

В корень проекта необходимо создать файл .gitlab-ci.yml в нём описываются 3 этапа:

- build
- deploy
- clenup

В даннном файле описан ci для company-service. Найти и заменить с company на {NAME_SERVICE}

> gitlab-ci.yml microservice

> gitlab-ci.yml micro-frontend

Нравится Оцените это раньше всех

Нет меток

1.0.10 Configuration parameters

Автор: Oleg Kravchuk
Последнее обновление: мар. 29, 2022

Micro-service configuration

Обычно в каждом проекте присутсвует 2 модуля это **Updater** и **Api**. В данном документе описывается общая конфигурация микро-сервисов. Конфигурация хранится в application.yaml в ресурсах модуля.

В проектах используется dotEnv и конфигурируется в файлах .env, в корне проекта присутсвует файл .env.example где хранятся значения:

```

DB_HOST=localhost
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=postgres

spring_profiles_active=dev # environment

```

› Api application.yaml

> Updater application.yaml

Это общие настройки сервиса, обычно они везде одинаковы:

- Подключение к базе данных, покдлючение кастомных функций.
- Настройка spring.
- Конфигурация aws s3, aws cloud.

Так же в проектах используется библеотеки briolink, для подключения, можно прочитать README.md каждой библеотеки.

Micro-frontend configuration

Для локальной разработки необходимо запустить проект.

Установка main-app

Он служит для точки входа для всех микро-фронтендов. При разработке фронтеда, этот проект должен быть всегда включен.

1. Склонировать репозиторий
2. Создать .npmrc из .npmrc_example и вставить NPM_TOKEN из variables
3. Собрать зависимости с помощью команды yarn
4. Настроить .env из .env_standlone, в нём находяться все URL до API микро-сервиса, URL до sumbodule и т.д.

Каждый модуль регистрируются в main-app, там же проходит конфигурация. В каждый модуль прокидывается информация о пользователе, JWT, URL модуля, правил роутинга, по global.

Конфигурация нового микрофронтеда

Исходные данные:

Название модуля	Company
URL модуля	`https://company.{ENVIRONMENT}.briolink.com`
URL API	`https://company-service.{ENVIRONMENT}.briolink.com`
Route	/profile/company/:slug, /profile/company/:slug/settings

Main-app

Для начала мы должны настроить main-app проект. Для начала нужно зарегистрировать микрофронтед.

В .env добавить URL module и URL API:

```
VITE_COMPANY_MODULE_URL="https://company.{ENVIRONMENT}.briolink.com"
VITE_COMPANY_API_URL="https://company-service.{ENVIRONMENT}.briolink.com"
```

Необходимо инцилизировать apollo client для company.

В src/configs/apollo:

```
const httpLinkCompany = createUploadLink({
  uri: import.meta.env.VITE_COMPANY_API_URL || process.env.VITE_COMPANY_API_URL,
});


export const companyModuleClient = new ApolloClient({
  ...defaultOptions,
  link: authLink.concat(httpLinkCompany),
  name: "company-module",
  queryDeduplication: false,
  defaultOptions: {
    watchQuery: {
      fetchPolicy: "no-cache",
    },
  },
  cache,
});
```

После нужно зарегистрировать модуль с помощью `qiankun`.

Значение	Описание
name	Уникальное идентификатор микро-фронтденда
entry	URL адресс расположение микро-фронтенда
container	Указатель на html элемент в который будет монтироваться микро-фронтенд

container	Указатель на html элемент в который будет монтироваться микро-фронтенд
activeRule	Правило роутинга, при каком будет просиходить рендер этого фронтенда
props	В данном случае, прокидывается stateData(global state) и apollo client для текущего микро-фронтенда

В src/main.ts:

```
registerMicroApps(
  [
    {
      name: "company",
      entry:
        import.meta.env.VITE_COMPANY_MODULE_URL ||
        process.env.VITE_COMPANY_MODULE_URL!,
      container: "#subapp-container-company",
      activeRule: ["/profile/company/:slug", "/profile/company/:slug/settings"],
      props: { stateData, apollo: companyModuleClient },
    },
  ]
)
```

После инцилизации, необходимо добавить в index.html вставить div id с значем container

```
      <div id="subapp-container-user"></div>

      <div id="subapp-container-company"></div>

```

```
<div id="subapp-container-connection"></div>
<div id="subapp-container-companyservice-service"></div>
<div id="subapp-container-search"></div>
```

Module

Для настройки модуля, необходимо в корне проекта создать файл vite.config.ts и добавить следующее:

```
const useDevMode = process.env.NODE_ENV !== 'production'
const baseConfig: UserConfig = {
    plugins: [
        vue({
            template: {
                /*
                  In the project custom components are used,
                  vue complains when compiling, in this case this
                  option allows you to prevent errors
                */
                compilerOptions: {
                    isCustomElement: (tag) =>
                        /^bl-/.test(tag) || tag === 'template-like',
                },
            },
        }),
        /*
            Init qiankun with id and debug mode
        */
```

```
20          qiankun('company', { useDevMode }),
21          // Init vue18n
22          vueI18n({
23              include: path.resolve(__dirname, 'locales/**'),
24              compositionOnly: true,
25          }),
26          svgLoader({
27              svgo: false,
28          }),
29      ],
30      resolve: {
31          alias: [{ find: '@', replacement: '/src' }],
32      },
33      build: {
34          rollupOptions: {
35              output: {
36                  /*
37                      Doesn't put packages in the shared bundle
38                  */
39                  manualChunks: {
40                      'vue3-apexcharts': ['vue3-apexcharts'],
41                      apexcharts: ['apexcharts'],
42                  },
43              },
44          },
45      },
46      server: {
47          fs: {
48              allow: [path.join(process.cwd(), '../../')],
49          }
```

```
        },
        port: 7106, // Each module must be on a unique port
        cors: true,
        hmr: {
            overlay: false,
        },
        base: '/',
    },
}

export default ({ mode }) => {
    Object.assign(process.env, loadEnv(mode, process.cwd()))

    baseConfig.base = process.env.VITE_BASE_URL
    if (mode === 'development') {
        baseConfig.base = '/'
    }
    return baseConfig
}

```

src/main.ts

```
let router: Router = null
let instance = null
let history = null

const i18n = createI18n({
    locale: 'en',
    legacy: false,
```

```
    messages,
})

function render(props) {
    const { container } = props

    history = createWebHistory('/')
    router = createRouter({
        history,
        routes,
    })

    instance = createApp({
        setup() {
            provideApolloClient(props.apollo)
        },
        render: () => h(App),
    })

    instance.use(router)
    instance.use(store)
    instance.use(i18n)
    instance.use(vClickOutside)

    instance.mount(
        container
            ? container.querySelector('#company-profile')
            : '#company-profile'
    )
```

```
}

/*  eslint no-underscore-dangle: ["error", { "allow": ["__POWERED_BY_QIANKUN__"] }]  */

if (!qiankunWindow.__POWERED_BY_QIANKUN__) {
    render({})
}

renderWithQiankun({
    mount(props) {
        render(props)
        instance.provide('$onGlobalStateChange', props.onGlobalStateChange)
        instance.provide('$setGlobalState', props.setGlobalState)
    },
    bootstrap() {
        // some logic
    },
    unmount(props) {
        instance.unmount(
            props.container
                ? props.container.querySelector('#company-profile')
                : '#company-profile'
        )
        if (instance.container) {
            instance.container.innerHTML = ''
        }
        instance = null
        router = null
        history.destroy()
```

```
66      },
67  })
68
69  // when run independently
70  qiankunWindow.customxxx = 'company-profile'
```

Нравится Оцените это раньше всех

Нет меток

1.0.11 Alerting, monitoring, logging

Автор: Oleg Kravchuk
Мар. 28, 2022

В данном документе описывается какие инструменты используется и на что обратить внимание, когда нужно отладить ошибку, либо посмотреть журнал действий микро-сервиса, нагружнность кластера.

На данный момент существуют 2 кластера:

- **dev-test-demo**
- **prod**

Logging

На dev-test-demo для логирования и мониторинга используется Kubernetes Dashboard.

Kubernetes Dashboard

Основные понятия:

Pods — это абстрактный объект в кластере K8S, который состоит из одного или нескольких контейнеров с общим хранилищем и сетевыми ресурсами, а также спецификации для запуска контейнеров.

Deployments — контроллер, который управляет состоянием развертывания подов, которое описывается в манифесте, следит за удалением и созданием экземпляров подов. Управляет контроллерами ReplicaSet.

ReplicaSet — гарантирует, что определенное количество экземпляров подов всегда будет запущено в кластере.

StatefulSets — так же как и Deployments, управляет развертыванием и масштабированием набора подов, но сохраняет набор идентификаторов и состояние для каждого пода.

DaemonSet - гарантирует, что на каждом узле кластера будет присутствовать экземпляр пода.

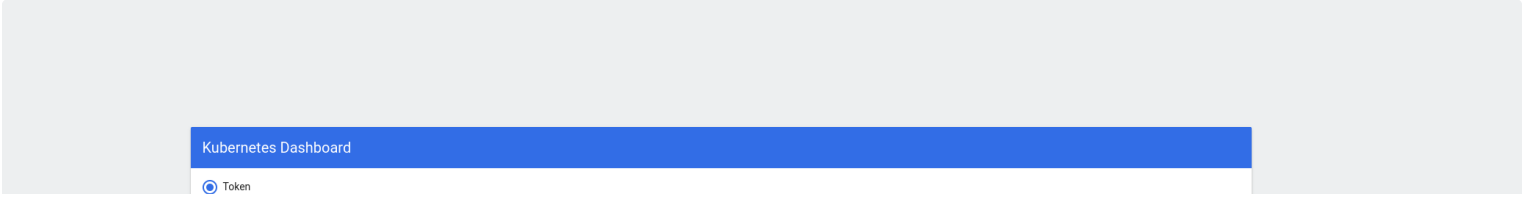
Jobs — создает определенное количество подов и смотрит, пока они успешно не завершат работу. Если под завершился с ошибкой, повторяет создание, которое мы описали определенное количество раз. Если под успешно отработал, записывает это в свой журнал.

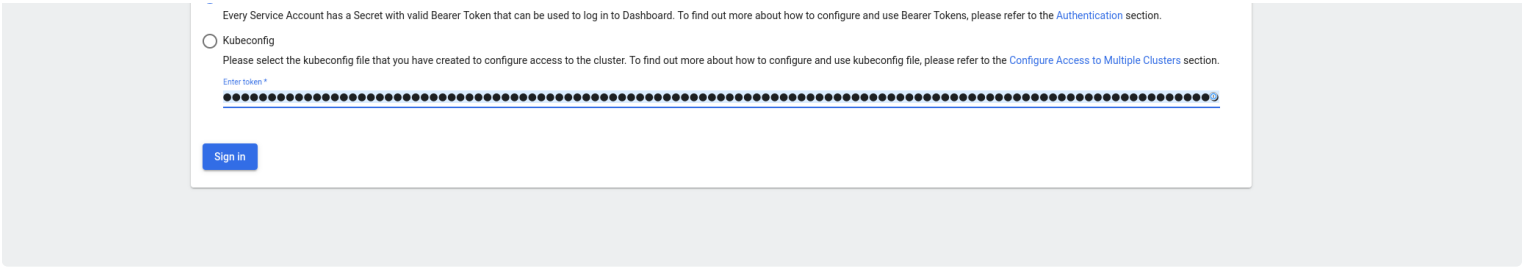
CronJob — запускает контроллеры Jobs по определенному расписанию.

Универсальный веб-интерфейс для кластеров Kubernetes. Он позволяет пользователям управлять приложениями, работающими в кластере, и устранять их неполадки, а также управлять самим кластером.

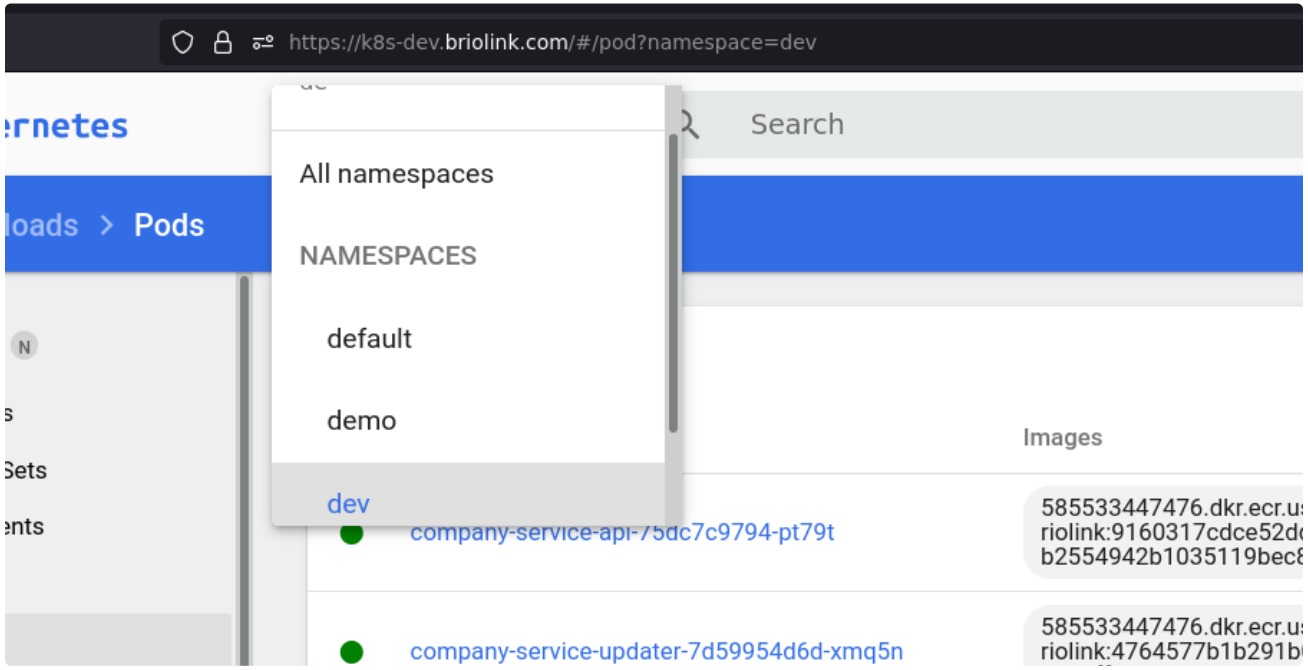
Для того чтобы войти, нужно знать токен.

Endpoint k8s на dev-test-demo — Kubernetes Dashboard





Каждый environment соотвествуют своему namespace



Чтобы просмотреть лог, необходимо зайти на environment → Pods → интересующий pod → нажать logs.

Для перезапуска микро-сервиса или микро-фронтенда необходимо перезпустить Deployment. Для этого необходимо зайти Deployments → интересующий deployment → нажать restart.

Monitoring

- Основной компонент — Prometheus. Prometheus получает метрики из разных сервисов и собирает их в одном месте.
- Grafana — отображает данные из Prometheus в виде графиков и диаграмм, организованных в дашборды.

Prometheus

Платформа для мониторинга, которая агрегирует метрики в режиме реального времени и сохраняет их в базу данных.

Endpoints:

- prod — https://prometheus.brielink.com/

- prod — https://prometheus.bnotlink.com/

Пароль можно найти в secrets manager, запись под названием “**prod/bl-network/infra**”

Для входа требуется:

- Username: monitoring
- Пароль: {prometheusPassword}

Graphana

Основные понятия:

- **Панель** — базовый элемент визуализации выбранных показателей. Grafana поддерживает панели с графиками, единичными статусами, таблицами, тепловыми картами кликов и произвольным текстом, а также интеграцию с официальными и созданными сообществом плагинами (например, карта мира или часы) и приложениями, которые также можно визуализировать. Можно настроить стиль и формат каждой панели; все панели можно перетаскивать на новое место, перестраивать и изменять их размер.
- **Дашборд** — набор отдельных панелей, размещенных в сетке с набором переменных (например, имя сервера, приложения и датчика). Изменяя переменные, можно переключать данные, отображаемые на дашборде (например, данные с двух отдельных серверов). Все дашборды можно настраивать, а также секционировать и фрагментировать представленные в них данные в соответствии с потребностями пользователя. В проекте Grafana участвует большое сообщество разработчиков кода и пользователей, поэтому существует большой выбор готовых дашбордов для разных типов данных и источников.
- В дашбордах можно использовать **аннотации** для отображения определенных событий на разных панелях. Аннотации добавляются настраиваемыми запросами в Elasticsearch; на графике

аннотация отображается вертикальной красной линией. При наведении курсора на аннотацию можно получить описание события и теги, например, для отслеживания ответа сервера с кодом ошибки 5xx или перезапуска системы. Благодаря этому можно легко сопоставить время, конкретное событие и его последствия в приложении и исследовать поведение системы.

Для того чтобы войти в graphana, нужно зайти на endpoint и авторизоваться через gitlab account.

Endpoints:

- prod — https://grafana.briolink.com

Чтобы посмотреть дашборд, можно выбрать один из несколько предложенных

Дашборд о кластере

Нравится Оцените это раньше всех

Нет меток

1.0.13 CI\CD

Автор: Oleg Kravchuk
Последнее обновление: мая 18, 2022

Определение

Непрерывная интеграция — это методология разработки и набор практик, при которых в код вносятся небольшие изменения с частыми коммитами. И поскольку большинство современных приложений разрабатываются с использованием различных платформ и инструментов, то появляется необходимость в механизме интеграции и тестировании вносимых изменений.

С технической точки зрения, цель CI — обеспечить последовательный и автоматизированный способ сборки, упаковки и тестирования приложений. При налаженном процессе непрерывной интеграции разработчики с большей вероятностью будут делать частые коммиты, что, в свою очередь, будет способствовать улучшению коммуникации и повышению качества программного обеспечения.

Определения использованные в документе

Название	**Описание**
Instance	Виртуальный сервер
Spot instance	Экземпляр, использующий свободные мощности EC2, которые доступны по цене ниже

	цене ниже
Stage	Определяет стадию, к которой относится задача;
Script	Действия, которые будут произведены, когда запустится задача
Tags	Теги, которые в свою очередь определяют, на каком раннере будет запущена задача
Only	Она определяет ветки, для которых будет создаваться пайплайн
Dependencies	Задачи, после которых нужно выполнить текущий job

Как это происходит?

После внесённых изменениях в only ветках выполняется pipeline. В котором определены этапы:

Build

Переда началом сборкой контейнера, ci request запрашивает у Gitlab Runner, который находится в instance, spot instanse, где выполняется инициализирование werf, обновление kubeconfig, после чего собирается docker container с билдом проекта. Собранный image публикуется в Amazon Elastic Container Registry.

- bl-network-prod-use2 — для prod окружения
- briolink dev-test-demo — для dev, test, demo окружений

Deploy

На данном этапе инициализируется werf, обновляется kubeconfig. Выкачивается собранный image.

Команда

```
1  werf converge --skip-build --set "env_url=$(echo ${CI_ENVIRONMENT_URL} | cut -d / -f 3)" --platform linu
```

Происходит деплой image в kubernetes с helm свойствами.

Clenup

В данном этапе происходит очистка кеша и неактуальных образов в container registry.

Examples

Настройка helm

- gitlab.ci.yaml - backend microservice

- gitlab.ci.yaml - micro frontend