FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Web interface for the deployment and monitoring of Nomad jobs
Student:	Bc. Pavel Peroutka
Supervisor:	Ing. Jaroslav Kuchař, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2019/20

Instructions

Nomad (by HashiCorp) is a container orchestrating tool for the deployment and monitoring of applications. Design and implement an open-source software solution for the self-service deployment and monitoring of Nomad jobs. The result product will be a web application (both frontend and backend) with a UI focused on straight-forward usage. UI is meant to be used by users without a deeper technical understanding.
This software is being created to be used internally by Ataccama company. It will be mainly used by the company's consultants to create new instances of offered software products to be used for proof of concept purposes.

1. Consult and define requirements with a client (Ataccama company).
2. Design an integration and implementation solution using Nomad. Select appropriate technologies for the project.
3. Implement the application.
4. Deploy and test the application in the client's existing DevOps process.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 25, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Web interface for the deployment and monitoring of Nomad jobs

Bc. Pavel Peroutka

Department of Software Engineering
Supervisor: Ing. Jaroslav Kuchař, Ph.D.

January 10, 2019

Acknowledgements

I would like to thank my supervisor Jaroslav Kuchař for leading me through the writing process of this thesis. My gratitude also goes to the Ataccama company and its employees, for giving me an opportunity to work on this project. Furthermore, I want to thank my great friend Tomáš for helping me with the English language. And finally, I would like to thank all of my numerous friends, my awesome roommates Kryštof and Prokop, my gorgeous fiancee Anežka and to my parents, for keeping me the best company I can imagine. Thank you!

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on January 10, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Peroutka, Pavel. *Web interface for the deployment and monitoring of Nomad jobs*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato diplomová práce popisuje návrh a vývoj alternativního uživatelskÃľho prostředí pro HashiCorp Nomad. Toto uživatelské prostředí je zaměřeno na uživetele, kteří nemají nutně zkušenosti s IT operacemi. Výsledný softwarový produkt, nazvaný Bedouin, je webová aplikace (obsahující server a klient), která je publikována jako open-source vydaný pod MIT licencí.

Klíčová slova HashiCorp, Nomad, Node.js, TypeScript, React, Fullstack, Webová Aplikace

Abstract

This master thesis deals with the design and development process of an alternative user interface (UI) for HashiCorp Nomad. This UI is meant to be used by users without deeper understanding of IT operations. The resulting product called Bedouin is a fullstack web application, published as an open-source under the MIT license.

Keywords HashiCorp, Nomad, Node.js, TypeScript, React, Fullstack, Web Application.

Contents

List of Figures

List of Tables

Introduction

Software development approaches are constantly evolving. The traditional waterfall model, which is still relevant for many cases, is moving away in favor of agile methods. Many companies are adopting cloud-based providing of their services, allowing them to quickly react to changing demands and increase the delivery rate of their products.

DevOps

From the high-level point of view, the software development life cycle consists of the development itself and the information technology operations (IT operations). The purpose of IT operations is to enable and support the functioning of infrastructure and the software running on it.

With the advent of agile approaches in software development comes the effort to highly automate IT operations. Agile development enables fast and flexible responding to requirement changes and empowers developers to release software more frequently. To be able to deliver releases in such a rapid manner, IT operations departments need to be well prepared for it, preferably by automating most of the routine processes. As a consequence of this need, a new field combining software development and IT operations for the sake of automatization was established. It is called DevOps.

HashiCorp Nomad

DevOps processes take various shapes depending on the specific requirements of a business. Common steps can be identified as: development, testing, package, provision, securing, deployment, and monitoring. HashiCorp company is one of many vendors offering tools for automating these steps.

Nomad by HashiCorp is a cluster manager and workload scheduling tool. It abstracts computing resources from the operator and thus decouples machines from the jobs to be done. [1]

Aim of the thesis

Nomad is controlled via the command-line interface. It allows the operator to submit a job that is described by a script written in Hashicorp's DSL [1]. Nomad also offers a graphical UI for monitoring the deployment of jobs.

Ataccama company is a supplier of various products focused on data management. One of Ataccama's Nomad use cases is a deployment of offered products for consultants, to be used for proof of concept purposes. The object of this paper is to create a self-service application with a user-friendly UI, both for submitting and monitoring the workload using Nomad. This application would allow the consultants to avoid coordination with the IT operations department, in order to provision the new instances of products they need.

This document will be publicly available as an open-source to be used by any other company or individual.

[1] Domain specific language

Chapter 1

Nomad introduction

In order a software application to run, it needs to be placed on a machine. The machine needs to have sufficient computing resources available. The application might have a specific requirements on the environment to be able to run properly, for example: specific version of the operating system, network access, Java Virtual Machine running, etc. In the case of a failure, it might be needed to run the application on another machine available, to provide high availability of the system.

The deployment of software applications to target machines can be handled manually, but in case of a bigger number of applications, deployment targets, or more frequent deployment interval, it can become a very time-consuming and tedious routine task. Nomad targets at automating such tasks.

Nomad is a cluster manager and scheduling tool. Scheduling, in context of Nomad, means a process of assigning workload to machines. Nomad scheduler makes placement decisions while satisfying specified constraints and optimizing resource usage of managed machines. It is designed to be able to handle thousands of scheduling events per second and tens of thousands of cores under management [2].

Nomad is an open-source member of HashiCorp's suite of tools for managing DevOps processes. The suite provides tools for setting up a development environment, packing applications, provisioning infrastructure, securing, scheduling deployments, and monitoring.

1.1 HashiCorp's suite of tools

Vagrant
: Vagrant helps developers to set up a development environment using virtualization. It allows developers to isolate their configuration and dependencies and share the same environment across all machines that are involved in the development process. For example, it allows members of a development team to use various operating systems for develop-

ment without encountering problems caused by running on a different platform. [3]

Packer
: Packer is a tool for building images for various target platforms.

Terraform
: Terraform allows treating a computer infrastructure as a code, using its high-level configuration language. It enables operators to keep track of different versions of an infrastructure, migrating the infrastructure to a new environment or moving back and forth between different versions. It is used to provision infrastructure both on in-house solutions or on IaaS[2] platform of many cloud providers. [4]

Vault
: Vault is an authentication and secrets management tool. It allows to centrally store, access and distribute secrets in a dynamic and scalable manner. It can be integrated with the scheduling process of Nomad. [5]

Nomad
: Cluster management and job scheduling are done with Nomad. Nomad decouples computing resources from workloads that are to be executed. This way developers are abstracted from the machines allowing them to avoid cooperation with IT operators.

Consul
: The last piece of the suite is Consul. Consul is responsible for service discovery and health checking of the running services. It can be used together with Nomad to register the scheduled tasks and run health checks on them. Then it can inform Nomad of failed processes and trigger restarting or reassigning of tasks. [6]

HashiCorp's products tackle DevOps delivery process on three distinct levels: the provisioning, security and run level. Where provisioning is managed by operations team, security by the security team and the run level is handled by developers (Table 1.1). These levels are loosely coupled, allowing the teams to work in parallel and abstract them from the rest of the process.

1.2 How Nomad works

Nomad forms an infrastructure by running Nomad agents on target machines. The agents are of two types: server and client. To submit a workload to Nomad, the workload needs to be declared by a script using a Hashicorp's

[2]Infrastructure as a service

Software delivery lifecycle	*Responsible team*	*HashiCorp tools*
Run Applications	Developers	Nomad, Consul
Secure infrastructure	Security	Vault
Provision infrastructure	Operators	Vagrant, Packer, Terraform

Table 1.1: HashiCorp suite of tools dimensions

DSL[3]. In Nomad terms, the workload is called "job" and the script declaration is "job specification". Nomad then performs an evaluation and decides an optimal placement on one of the client machines.

1.2.1 Nomad job

Nomad job consists of a set of tasks, that can be placed together in task groups. A task represents elementary unit of work, for instance: starting a Docker container, executing a binary, running a Java Jar file, etc. Tasks that need to run together are placed in the same task group. Nomad then ensures to run them on the same machine.

Nomad job is specified by structured, both human and computer readable language, defined by the HashiCorp company, called HashiCorp configuration language (HCL). The general structure of a job is: job → task group → task.

Nomad identifies a job by its name. A job specification having a name that already exists in a different job is treated as another version of that job. A job can be updated this way, on the fly, without the need to kill the old one and recreate it all over. Nomad comes with rich functionality to handle updates, such as blue/green upgrade or canary upgrade.

Job can be of three types: service, batch and system. The type instructs Nomad what kind of scheduler to use. [7]

Service
: The service type is meant to be used by long-running jobs, such as a web application. In such case, Nomad takes extra time to find the best-fit target machine to run the job.

Batch
: The batch type denotes short-lived jobs. Example of a short-lived job can be periodic database backup. This type of job is not that sensitive to occasional performance bumps. This is, again, taken into account by Nomad in order to find the target node.

System
: System job should be running on all the clients managed by Nomad

[3]Domain specific language

that meets the job's constraints. So when a new node joins the Nomad cluster, the job gets rescheduled, so it is allocated on the new member as well. This type of job can be used for example for logging services.

Job specification

HashiCorp configuration language defines short building blocks called job stanzas, of which the job specification is composed. HCL defines more than 20 stanzas. Each stanza has a focus on a distinct part of the configuration, for example setting resource parameters required by task or restart strategy of a task group. There is a job specification example shown in Listing 1.1, that describes deployment of Redis (a lightweight in-memory database).

```
job "redis" {
        datacenters = ["dc1"]
        type = "service"
        update {
                auto_revert: true
                max_parallel = 1
        }
    meta {
        greeting = "Hello CVUT!"
    }
        group "cache" {
                count = 2
                restart {
                        attempts = 2
                        interval = "1m"
                        mode = "fail"
                }
                task "redis" {
                        driver = "docker"
                        config {
                                image = "redis:latest"
                                port_map {
                                        db = 6379
                                }
                        }
                        resources {
                                cpu = 500 # 500 MHz
                                memory = 256 # 256MB
                                network {
                                        mbits = 10
                                        port "db" {
                                        }
                                }
                        }
                }
        }
}
```

Listing 1.1: Nomad job specification example

Stanzas used in Listing 1.1

`job`
: The job stanza is a top-level stanza. The stanza itself describes the overall behavior of a job, job name, type, priority, target datacenter, etc. It is a container for task groups.

`update`
: The update stanza defines the update strategy and its parameters, such as minimum healthy time to consider update as stable.

`meta`
: The job, task group and task stanza can contain arbitrary user-defined data in a key-value shape.

`group`
: Tasks are arranged in the task group stanza. The task group can define a number of task group instances that are to be scheduled, constraints that need to be satisfied on a target machine, etc.

`restart`
: The restart stanza tells Nomad how to behave in case of a task group failure. How many restarts are allowed and in what interval should they take place.

`task`
: Task is the core stanza describing what should be executed. There are a few kinds of drivers that a client uses to execute the task.

`resources`
: The resources stanza declares resource requirements of a task. The stanza involves CPU, iops[4], memory and network resources.

1.2.2 Job update

When a specification of an existing job gets changed and submitted to Nomad, few scenarios can happen depending on a specified update policy (by the update stanza).

When an update policy applied, Nomad scheduler plans a so-called "deployment". Deployment can be thought of as a single transaction for all the planned changes. Very much like a transaction in RDBMS[5] can be rollbacked or committed, Nomad deployment can be reverted back in case of failure. Respectively, when a canary update applied, the deployment can be manually promoted after it is made sure, that everything is working as expected.

[4]Input/output operations per second
[5]Relational database management system

When no update policy is applied, the update happens without using a deployment. Then there is no control of ongoing update and the job cannot be reverted to the previous version.

Rolling upgrade

The rolling upgrade strategy is a straightforward way to perform an upgrade. When this strategy used, Nomad upgrades a specified number of task group allocations at the time. That means it creates a specified number of new allocations and takes the same number of old allocations down. The number of parallel upgrades is `max_parallel` parameter of the update stanza. The update stanza also defines the following parameters:

`min_healthy_time` This parameter specifies minimal time interval that the allocations has to be successfully running, before proceeding to upgrading other allocations.

`healthy_deadline` Maximum time to mark allocation as healthy. When the deadline is not met, the deployment is pronounced as failed.

`auto_revert` Specifies whether the deployment should be auto-reverted in case of failure, or should be handled manually.

In case of failure of any of the new allocations, the error is propagated, allocation of new instances is stopped and the deployment is stopped. If `auto_revert` parameter is on, then the system automatically ends up in the previous state as it was before beginning the upgrade. [8]

Canary upgrade

The rolling upgrade and the canary upgrade strategies are very much alike, except the canary upgrade takes one extra step, to ensure that the changes are working properly. In this case, the update stanza specifies parameter `canary`, telling Nomad the number of allocation of the new version to be deployed, while persisting all the old allocations. After it is made sure, that the canary allocations are good to go, the deployment has to be manually promoted. When promoted, the deployment continues in the same manner as the rolling upgrade. [9]

Blue/green upgrade

Blue/green upgrade uses the canary concept from the previous strategy with a little modification. When a canary deployment is considered safe, all the rest of allocations is upgraded at once. As opposed to the rolling upgrade, which upgrades the given number of allocations at once. [9]

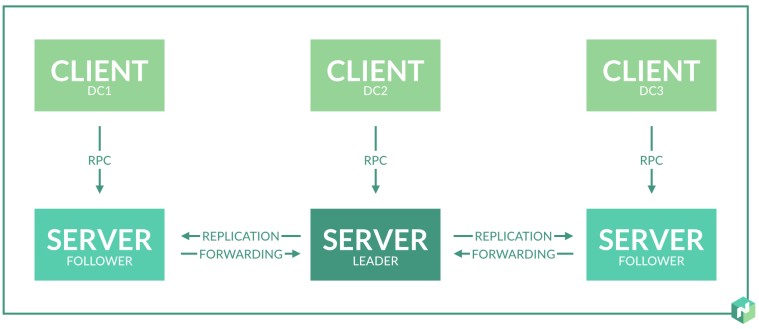

Figure 1.1: Nomad region architecture

Source: https://www.nomadproject.io/

1.3 Nomad architecture

Nomad infrastructure consists of Nomad agents that runs either in the server or client mode. These agents are located on machines forming datacenters and eventually regions. An example of a region structure is shown on Figure 1.1

A datacenter is made of a server and clients. The clients registers themselves to a server and provide all the information necessary for allocation decisions to the server. The information includes available resources, installed drivers, installed operation system, and other attributes. A client also provides the server with information about the status of its allocations. The server of a datacenter processes requests coming from the leader server of a region and takes case of the allocation decision process within the datacenter.

By default, Nomad creates a single global region, that contains all the datacenters. Servers of a region forms a cluster and replicates data between each other, in order to ensure high availability. Within the cluster there is one server elected as a leader, that is responsible of handling all requests and queries.

Regions are independent from each other, each running its own clients and its own jobs. Nomad, however, supports federating regions together, forming them into a single cluster.

A client is a target machine where the tasks are executed.

A server manages communication within a cluster of servers, runs data replication and does the placement decisions of tasks.

Regions and datacenters forms an overall structure of Nomad environment. Regions are sets of datacenters and can be federated together. Nodes in a datacenter are expected to share local network.

1.4 Scheduling

Scheduling is a decision process of assigning tasks to client nodes. Result of the process is an evaluation. Evaluation describes mapping of task groups to clients, and the actions needed to reach the desired state. Scheduling process is handled by server nodes and must respect task requirements specified by job specification.

Entry point to scheduling process is submitting a job specification to a Nomad server. Scheduling also happens in other situations when the current state does not match the desired state, such as updating the job specification, stopping the job, or failure of allocated resources.

Allocation decision takes two steps: feasibility checking and ranking. The first step picks nodes having the necessary capabilities to run specified tasks, such as the right set of drivers or enough of computing resources. Subsequently, the scheduler ranks the resulted node in order to select the best fit. [10]

Dictionary

Allocation
: An allocation is a mapping of a task group instance to a client node, which resulted from scheduling.

Evaluation
: An evaluation is an entity representing Nomad's scheduling decision. It is a plan of placing allocations to reach desired state.

Deployment
: A deployment is a transactional step of allocating new resources.

1.5 Consul integration

Consul is a tool for discovering, configuring and monitoring services in a given infrastructure. It helps to keep dynamic infrastructure organized and well performing. Consul is a tool of the HashiCorp's collection and is closely related to Nomad functionality, even though is not required to use it.

Service discovery

Consul has a service registry that allows it to collect and store information about running services and providing the information to its clients. An example of such service could be a PostgeSQL server or a web application.

Nomad provides a way to integrate with Consul services by introducing `service` stanza of the Nomad job specification. The service stanza specifies service attributes, such as name, port, tags, etc.; and health check parameters,

describing a way how to check whether the service is alive. Running a Nomad job with the service specification will automatically registers the services to Consul.

Health checking

Consul supports running health checks using HTTP, TPC and gRPC protocols, or using a bash script. There can be multiple heath checks running on a single service. All these parameters can be specified using `service`.

Consul's health checking capabilities are very useful for Nomad's allocation and deployment management. Nomad itself checks for healthiness of its allocations, but without specific health checks set up in place, it can check only for system-level errors, such as running out system resources or network failure. It would not be able to detect for example a problem, when an allocation is placed and running properly, but the process inside the allocation is not able to start, due to some specific error, and is constantly restarting.

Nomad benefits from Consul integration by using the health checks to mark allocations as failed, when its service has failed. This feature is utilized both for allocation failure management when the job is already running, as well as for managing job updates.

Key-value store

Next Consul's feature is a hierarchical key-value storage. It is a general purpose storage, typical usage is for including dynamic configuration, feature flagging, coordination, leader election, etc. [6] The Consul clients can easy retrieve and use this data.

Consul introduces a utility tool called Consul Template. This tool takes an template file with template variables defined and interpolates[6] the variables with values, using a Consul key-value store, Consul's environment variables, or Vault[7] variables.

Nomad job can take advantage of Consult Template for configuring its tasks. The job specification defines `template` stanza, which is used to declare a template for Consult Template to use. The then stanza says where should the interpolated template be places in the allocation's environment and how should the process be informed in case of template change.

[6] Interpolation (in computer programming) is a process of evaluating literals in order to
[7] Vault is HashiCorp's secret management tool

Chapter 2

Project domain

The output of the practical part of the thesis is an alternative user interface for Nomad. This software is being created at the initiative of the Ataccama company. There is a particular use case the company wants to tackle — to allow Ataccama's consultants to be able to provision new installations of offered products without the need of coordination with the IT department (further described in subsection 2.1.2). This new user interface will be realized as a web application. It will be tailored for the consultants as the target users, providing them with straightforward, uncomplicated self-service Nomad interface. While this is true, the application will be available for general use by everyone for similar use cases. The application will be called **Bedouin** and it will be further referenced by this name within this document.

2.1 Use of Nomad at the Ataccama company

2.1.1 Continuous delivery

While developing a big software product, especially using the agile methodology, the source code is often being written by several teams at the same time. In such case, the source code is often branched into multiple branches, potentially at different stages of the development. If the product is composed of multiple parts, such as frontend and backend part of a web application, it adds another level of complexity to the development process. For example, when a specific backend branch of code needs to be running on a development server in order to develop the frontend part accordingly, there needs to be a well-organized process in place. Another case would be running a development version of a product with a significant amount of changes, in order to be tested before merging it to the main branch of the source code.

A common solution for this situation is setting up a continuous delivery process. Continuous delivery is an automated series of steps that are applied to a software source code. It is triggered by the submission of a new version

of a code into a versioning system. This series of steps is called a "pipeline". There is a number of tools to provide a framework for building the pipeline. [11]

The Ataccama company uses Jenkins for its continuous delivery processes and Nomad takes part in some of them. During the process, after the source code is grabbed by Jenkins, built, tested and packed, the last step of the pipeline is up to Nomad and Levant[8]. Jenkins invokes the Levant using its command line interface, passing the template file (that is prepared in advance) along with all the template parameters to it. The template parameters usually involve version numbers of the software used, datacenter that should be used for the deployment, or the name of the Nomad job. After Levant renders the job specification with given parameters and passes it to Nomad, Jenkins then reports the result status of the deployment.

The logs of the deployed applications can then be observed in multiple ways: using the Hashi UI application (running in the company's network); via the Nomad command-line interface; or browsing Ataccama's Kibana [9].

Ataccama also runs Sensu — a scalable monitoring framework. In this case it is used to monitor resources of machines that are running the deployed Nomad job allocations.

2.1.2 Production

The second Nomad use case at the Ataccama company takes place during presale of Ataccama products. As a part of a product offering to a potential customer, so-called "proof of concept" (PoC) phase happens. During this phase, An Ataccama's consultant demonstrates the capabilities of the product to a customer, in order to present features that a customer can benefit from, so that the customer would purchase the product. Before the proof of concept happens, a consultant needs to have a fresh new installation of the presented products. The installation is usually deployed in a cloud environment, so it is easily accessible over the internet.

In the current situation, when the consultant needs to be provided with the new installations, he/she assigns a task to the IT department to deploy the new instances of the products needed. At that point, the IT department has already set up the cloud infrastructure with the Nomad in place. It also disposes of all the Nomad jobs specifications of desired products and its dependencies, such as databases, proxies, load balancers, etc. The IT department needs to submit the prepared templates with the desired parameters to Levant and ensure that the deployment is successful. Once it is deployed, the consultant sets all the necessary customer-specific configurations of the product. To do so, the consultant needs to be able to browse the applications logs. When the configuration causes an error, the logs provide the consultant with

[8] Templating and deployment tool for Nomad, further described in section 2.3

[9] Kibana is a framework used to build centralized logging database

an explanation of what went wrong. Currently, in order to observe the logs, they need to use an SSH connection to the target servers.

2.2 Product requirements

There are three major product requirements. First, the consultants should be abstracted from the Nomad job specification and at the same time they need to be able to modify specific parameters of the job. For example, a version number of the offered product that is about to be deployed. Second, they need to be able to monitor the status of the deployment, to see whether it was successfully deployed or if there was a problem they need to consult with the IT department. And lastly, they have to have access to logs produced by deployed applications.

2.2.1 Templating

To achieve the first requirement mentioned, the Nomad job specification needs to be parameterized. Nomad implements a feature to parameterize job specification by dispatching variable values along with the job specification when submitting the job. These variables get then interpolated by Nomad, but it only applies to some of the stanzas of a job specification. To provide greater flexibility of the specification, additional templating functionality needs to be used.

At Ataccama, templating capabilities of the Levant tool is being currently used for parameterizing Nomad job specification. Thus, compatibility between templating mechanism of Bedouin and Levant is desired.

2.2.2 Ease of use

Besides the command-line interface, Nomad provides a simple web application for monitoring the submitted jobs. It displays detailed information about the job itself and about the scheduling process and all the entities involved in the process. That is useful for DevOps operators, but it is not very user-friendly for the consultants.

2.3 Prior art

There are quite a number of Nomad tools available, either enhancing the functionality of Nomad or providing integration of Nomad with other DevOps tools. Tools implementing at least some of the requirements imposed on this project are: Nomad web UI, Hashi UI, Levant, and Cluster Broccoli.

2.3.1 Nomad web UI

Nomad is equipped with a web user interface. It is mainly designed for job monitoring, as it has very limited capabilities of controlling the jobs. The web UI provides advanced information about the running jobs, as well as information about Nomad servers and clients in the cluster. Using the web UI, it is possible to view all the job evaluations; job versions and diffs between each one; or task group allocations, down to the level of the client logs of a particular task. The UI also handles displaying deployments and its status, such as running canaries, etc.

2.3.2 Hashi UI

Hashi UI is a third party web application offering more sophisticated UI over Nomad web UI. It displays Nomad state information in greater detail and comes with more advanced features, such as real-time resource utilization monitoring dashboard and previewing and downloading files placed on a client within an allocation. Hashi UI also gives user greater control over the job.

- The user is able to edit the job specification of the running job and re-submit it.
- It is possible to increase or decrease the count of task group allocations on the fly, just by clicking a button in a job view.
- Canary deployment can be easily promoted when the canary or the blue/green upgrade strategy is used.

2.3.3 Levant

Levant is a command-line tool adding an extra layer to Nomad. Levant adds templating capabilities to Nomad by introducing extra parameters to the job specification. Levant provides real-time feedback of job scheduling and deployment, as well as detailed description of errors. It also provides extra functionality to the Nomad deployment by supporting auto-promoted canary deployments and tracking of failed deployments that resulted in a rollback to the previous job version.

Templating

Levant provides Nomad job specification files with a free-form templating capability. By virtue of this a special parameter placeholder syntax is defined, which can be used throughout the job specification. Levant template syntax involves specifying parameters that are resolved by user-provided data; parameters that are resolved by Consul's key-value store; and even logical expressions and functions.

This templating mechanism has no limits in the sense of what portion of a template can be parameterized and by what value it can be replaced. It thus gives much more flexibility than the templating feature of Nomad.

User-provided template parameters
This is a simple kind of parameters that get interpolated by values provided by the user. Parameter syntax uses a dot notation[10] to address the value, it can therefore be resolved by a hierarchical object (described by JSON or YML). An example of a parameterized `resources` stanza and the corresponding parameter values object is shown below (Listing 2.1 and Listing 2.2).

```
resources {
 cpu=[[.res.cpu]]
 memory=[[.res.memory]]
 network {
  mbits=[[.res.network.mbits
      ]]
 }
}
```

Listing 2.1: Parameterized stanza example [12]

```
{
 "res": {
  "cpu":250,
  "memory":512,
  "network": {
   "mbits":10
  }
 }
}
```

Listing 2.2: Parameter values example [12]

Template functions
Levant implements a number of build-in functions that can be used to resolve the template parameters.

One of the functions — `consulKey` is able to retrieve data from Consul's key-value store. It takes an argument specifying the key of the Consul store record, in order to use its value to resolve the parameter (Listing 2.3). [13]

```
[[ consulKey "service/config/cpu" ]]
```

Listing 2.3: ConsulKey template parameter [12]

Finally, Levant template parameter can also be an expression. Example code is shown at Listing 2.4.

```
{{ if consulKeyExists "service/config/alerting" }}
  <configure alerts>
{{ else }}
  <skip configure alerts>
{{ end }}
```

Listing 2.4: Expression template parameter [12]

[10]Dot notation describes a path to an object property by using dots for separating the property name from its parent object name.

Parameter declaration file
In order to specify template parameter properties, such as name or default value, it is possible to declare it in an additional file. Levant accepts parameter declaration specified in the Terraform[11] variable configuration format (Listing 2.5). Even though it is not the optimal way to specify the template parameters, Levant uses it for the sake of consistency with the HashiCorp's suite of tools. [12] Terraform's variable configuration allows to specify a parameter type, human-readable description, and a default value.

```
variable "resources_cpu" {
  description = "the CPU in MHz to allocate to the task group"
  type        = "string"
  default     = 250
}

variable "resources_memory" {
  description = "the memory in MB to allocate to the task group"
  type        = "string"
  default     = 512
}
```

Listing 2.5: Example of Terraform variables [12]

2.3.4 Cluster Broccoli

Cluster Broccoli is a minimalist Nomad web interface. Its goal is very similar to what the Bedouin project is targeting — a self-service Nomad interface to be used by users without deeper technical knowledge. According to its documentation: "Cluster Broccoli is meant to be setup by your IT. Some technical knowledge is required to setup the infrastructure and define the templates. End users can be internal (QA, data scientists) or external (customers, potential customers)." [14]

Cluster Broccoli templating feature is very much the likes of Levant's parameter substitution with user-provided data, except that it comes with its own template parameter syntax. It can work only with the JSON job specification, so it cannot benefit from good human-readability of HashiCorp configuration language. Cluster Broccoli comes with its own proprietary format to declare template parameters, which is much more suited for its purpose than the Terraform language (the case of Levant). Cluster Broccoli supports a wider variety of parameter data types. It also lets the operator specify a description of the purpose of the job template as a whole and human-readable parameter labels.

[11] HashiCorp's infrastructure provisioning tool.

The user interface of Cluster Broccoli allows a user to input the template parameters via a form when submitting a job. It provides the user with real-time feedback of allocation statuses and allows them to browse logs of the running tasks.

Chapter 3

Analysis and design

Software requirements analysis and designing the application is an essential part of the software engineering discipline. It is a phase that precedes the actual development of the application. Its main purpose is to refine and formulate the requirements that came up during the requirements gathering. The resulted documentation should be agreed on with the client and used for the design process that follows. In the case of agile development, the process of requirement gathering and refinements occurs periodically as the new requirements rise in time.

In the case of Bedouin, the software process follows the waterfall model to the point of the complete first version, as described further in this chapter. Once the initial phase is completed and published under open-source license, further development will be available for anyone to join. Either by requesting new features, reporting found bugs, developing, evaluating, or reviewing the code changes. The project will be free to be forked by any party interested, who can subsequently implement their own specific functionality.

Bedouin's requirements were gathered during several meetings with the lead DevOps engineer of Ataccama. The members of the Ataccama DevOps department are currently the ones handling the process of provisioning the software for the consultants. Bedouin should eliminate most of the manual coordination between the consultants and the DevOps department.

3.1 Functional requirements

Functional requirements define what the demands placed on the software product concerning its performative capabilities are, how the software should behave and what it should be able to execute. It is a result of the requirement gathering process. Functional requirements define the scope of the project, it can be viewed as a checklist to be used while developing.

3.1.1 Read template and template parameters files

The application will be able to read template-related files from a file system. The location of template files will be defined in the application configuration. Both the template files and the template parameters files will be compatible with the Levant templating syntax.

3.1.2 Render templates

Next feature required is template rendering. The final product is expected to implement at least the functionality of the template parameter substitution with provided values. The rest of the Levant templating features is optional.

3.1.3 Submit job to Nomad

Bedouin is required to provide the user with an interface for inputting the template variables. Subsequently, Bedouin has to be able to submit the job to Nomad and continuously inform the user of the status of the deployment process.

3.1.4 Update job by its template

The user needs to be able to update the job that is already running by editing the template parameters of the template the job originated from. If the template files are no longer by available, this option will not be shown.

3.1.5 Stop the job

The user interface has to enable the user to stop the job execution.

3.1.6 Audit information about a user

The system has to keep track of the users who submitted a job.

3.1.7 Monitor ongoing job

After the job submission, or update, Bedouin's user interface must provide information about the job progress status; allocations state; and information about the Nomad clients on which the allocations are running.

3.1.8 Read task logs

The user has to have access to the system log that is produced by the process of a task within an allocation. Both the standard output and the standard error output (`stdout`, `stderr`) are needed.

3.2 Non-functional requirements

Non-functional requirements, also called general requirements, represent a set of constraints regarding the qualities that the system has to satisfy. In simple terms, it is not a description of what the system should do, but rather how the system works and how it is implemented (as opposed to the functional requirements).

3.2.1 TypeScript and React

Part of the Ataccama's development department is focused on web application development using the TypeScript language and the React framework. Both TypeScript and React are widely used and popular technologies with large developer community [15]. Since the application will be open-sourced, it is desired to pick popular, well-known technologies. It is supposed that Ataccama employees will be able to participate in maintaining the project. The use of TypeScript and React is hence declared as a requirement.

3.2.2 OpenID Connect authentication

Authentication via the OAuth 2.0 standard and the OpenID Connect protocol is used across most of the systems in Ataccama. The application is expected to provide a way to integrate it with an OpenID Connect authentication provider. Only the authenticated users shall be able to deploy jobs and also the authentication would be used for identifying the user for audit. This feature will be flagged[12], allowing to use Bedouin with no OpenID provider.

3.2.3 Dockerized

The application will be dockerized for the sake of portability and easy deployment.

3.2.4 Stateless as possible

The application should hold as little state due to simplicity. All the job-related state should be left to be handled by Nomad and the job templates shall be stored in a file system. Nomad provides a way to store key-value shaped data at the job, task group, and task specification. Bedouin should take advantage of this feature in order to keep the state at the minimum.

[12]Feature flagging is a mechanism allowing to change application's behavior by disabling and enabling its features.

3.2.5 Provide logs

The application should provide logs to be integrated with the Ataccama's central logging system — Kibana.

3.3 Stakeholders

The Bedouin application will serve the purpose of separating concerns of IT operators and "ordinary" users (in the sense of users without deeper technical understanding).

IT operators will most likely interact with the application only at the beginning while it is being installed and configured to work with a Nomad instance. Their job is to prepare and maintain template files to be used by the end users. In case of a failure of any kind, IT operators can use Bedouin as one of the tool for troubleshooting, either through its UI or by inspecting the output logs.

The users of Bedouin will interact with the application after it is fully set up. They will be able to schedule a workload as easily as filling out the template parameters and hitting the submit button. Since Nomad scheduling is a complex process, during which various kinds of problems may occur, it is assumed that there will always be an IT support to help with troubleshooting. An error may occur even when the user has done everything correctly. The user is not expected to deal with Nomad-related errors, rather, they are expected to hand over the problem to the IT support. As a result, it is possible to use Bedouin even as an inexperienced user.

3.4 Domain model

Bedouin's domain model is based on the model of Nomad. It can be considered as an extension of it. Nomad's domain model is not a part of the Nomad documentation that is publicly available. The shape of the Nomad entities and relationships between them, as illustrated in figure 3.1 are reconstructed from the shape of the objects that are being served by the Nomad API. The model is not meant to be considered an official domain model of Nomad.

The part of the model describing Nomad entities is viewed from Bedouin's perspective and is intentionally simplified. It displays only the entities that are relevant to Bedouin's functionality and also relationship cardinalities are reduced in some cases. For example, the relationship between `JobVersion` and `Evaluation` entities is in fact one-to-many, but Bedouin is interested only in one particular `Evaluation` per `JobVersion`, resulting the relationship to be reduced to one-to-one.

The model shows description of the Bedouin template-related entities and how these entities relate to the Nomad model.

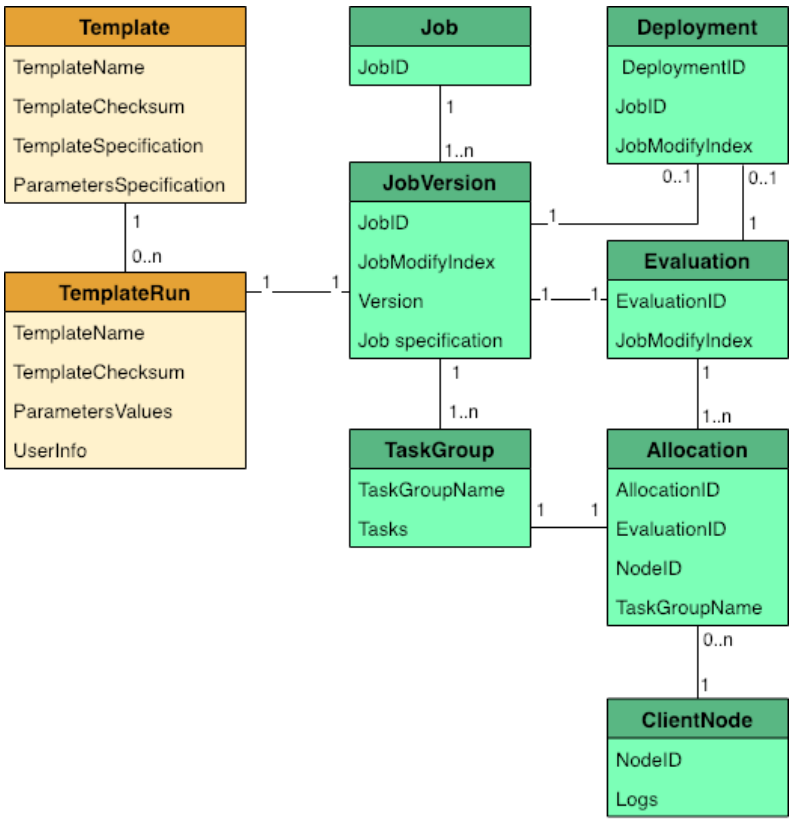

Figure 3.1: Bedouin domain model

Description

`Job`, `JobVersion`
: One instance of a Nomad job specification that defines a job name corresponds to one version of a Nomad job of that name. So, when there are two job specifications with the same job name submitted to Nomad, they are considered to be two distinct versions of one single Nomad job[13]. To model this concept, there is a Job entity defined that is a parent entity of all its specifications.

`TaskGroup`
: The `TaskGroup` entity represents the `group` stanza of the job specification. This stanza is materialized and explicitly described as a domain model entity in order to form a relationship with the `Allocation` entity.

[13]Even if one job specifications gets submitted to Nomad multiple times, without any change, each submission creates new job version.

`Deployment`, `Evaluation`, `Allocation`, `ClientNode`
: These entities are direct representation of concepts described in section 1.2.

`Template`
: This entity stands for a template version, which is defined by the template specification and the template parameter specification files. A versions of a templates are distinguished by a checksum (hash value[14]) of its template files.

`TemplateRun`
: The `TemplateRun` entity represents an act of a template render and the following submission to Nomad. This entity carries the parameter values that have been used for the rendering and information about the user that had run the template.

  A relationship with `Template` is made by the `checksum` property. If the template gets changed or deleted, `TemplateRun` is not be able to reach it. That is the reason for the optional cardinality on the side of `Template`, which may seem confusing at first. Bedouin has no control over template files since they reside on a file system.

  A relationship with `JobVersion` is optional as well. Bedouin is displaying all Nomad jobs no matter if they were scheduled via Bedouin or not.

3.5 Architecture

The Bedouin application consists of two essential parts — a server (backend) and a client (frontend). The server part does the heavy lifting: handling template files; template management; executing the flow of submitting a job to Nomad; and all the communication with Nomad. Whereas the client part's purpose is to provide the user with a graphical interface in order to control the server part.

3.5.1 Backend

The Bedouin server can be viewed as an addition layer to Nomad, using a specific part of its API and enhancing it by extra functionality.

Template files There are two kinds of template-related files: the template specification files (parameterized Nomad job specification) and the parameters declaration files. Template files will be organized in the following structure: each pair of the template specification and parameter declaration will be

[14]Hash value is a fixed-size image of arbitrary data, it is a result of a hash function.

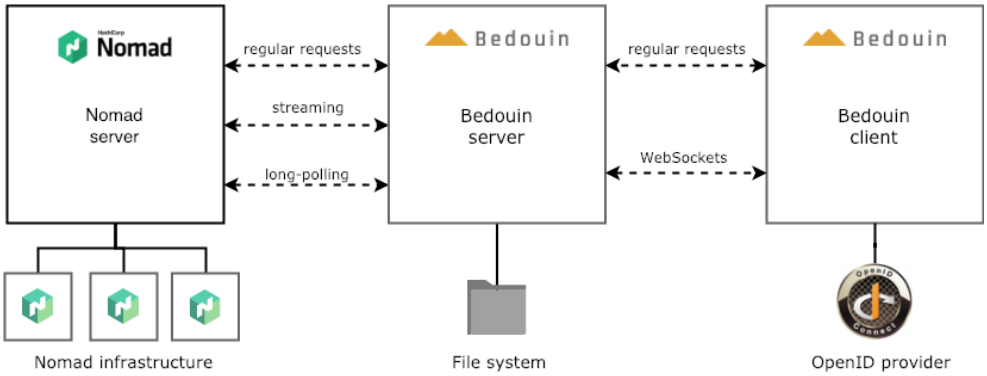

Figure 3.2: High-level Bedouin architecture

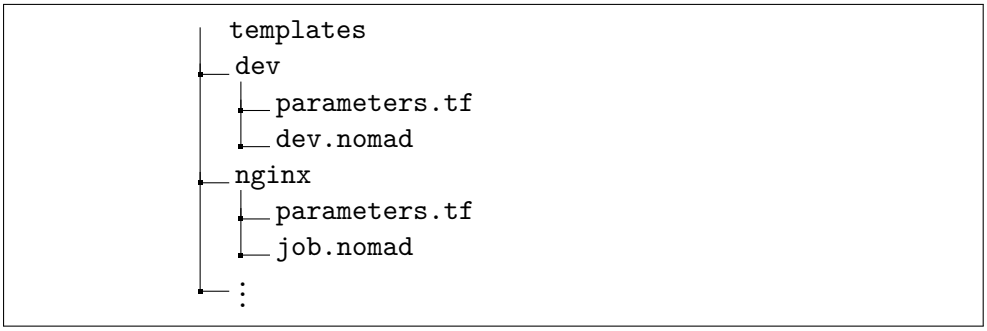

Figure 3.3: Template files structure

placed in its own folder and these folders will reside together in a single root folder. An example of this structure is shown in Figure 3.3. It will be within the competence of the Bedouin server to handle and manage the template files.

Running the templates The server is responsible for handling the process of running the templates, which involves rendering the template and submitting it to Nomad.

The job-related state is handled by Nomad, the only Bedouin-related data is an audit of the user actions. Due to the minimal state requirement, is it agreed that it is sufficient to store only the user information of the job submission. It is thus possible to omit usage of persistent storage and use the job's `meta` stanza allowing to store arbitrary key-value shaped data.

Proxying and transformation of Nomad API Nomad can be fully controlled by its RESTful HTTP API. Due to the fact that Nomad is a long-running service with dynamic environment, it is often needed to observe the API resource over time to inform the operator of change as soon as it happens.

Nomad allows to continuously query its resources by using the long-polling technique. The Nomad long-polling is further described in subsection 5.1.3.

The Bedouin server will be handling all the Nomad API communication and the data concerning job monitoring will be forwarded to the Bedouin client. The API will be reshaped to match the requirements of the Bedouin application. In order not to have to replicate the long-polling technique and at the same time leveraging the benefits of persistent connection using WebSockets, the resources that need to be observed over time will be forwarded to the client using the WebSockets technology.

```
/jobs
/job/:job_id
/job/:job_id/evaluations
/job/:job_id/deployments
/job/:job_id/allocations
/deployment/:depl_id/allocations
/evaluation/:eval_id/allocations
/allocation/:alloc_id
/client/allocation/:alloc_id/state
/client/fs/logs/:alloc_id?task=<task_name>
```

Listing 3.1: Part of Nomad API

```
/jobs
/jobs/:job_id
/jobs/:job_id/spec
/allocations?(job=<job_id> | deployment=<depl_id>
                           | evaluation=<eval_id>)
/allocations/:alloc_id
/allocations/:alloc_id/stats
/allocations/:alloc_id/logs/:task_name
```

Listing 3.2: Bedouin API

Authentication Bedouin has to provide a way to integrate it with the OpenID Connect authentication provider that is used in the Ataccama company. Ataccama uses solution based on Keycloak[15]. Though OpenID Connect is a widely known protocol, it should not be assumed that every party using Bedouin in the future will use this feature. In order to keep the application generic, the authentication feature will be feature flagged. If needed, this feature will be implemented separately in a distinct feature branch of the code.

3.5.2 Frontend

Frontend part will implement an API client to communicate with the Bedouin backend and the user interface based on the React framework. It also needs

[15] An open source identity and access management framework

to implement the corresponding part of the OpenID Connect protocol communication flow.

3.6 Summary

The product name will be **Bedouin**. Bedouin will be an alternative Nomad interface implemented as an additional layer on top of Nomad API. It is designed to be used by users without deeper technical knowledge (non-developers, non-IT-operators), allowing them to operate Nomad jobs. It is not meant to replace Nomad tools currently used by IT operators, providing them with an interface with greater control over the system.

Bedouin will be a web application composed of the server and the client part. The server part will be responsible for handling Nomad job templates and communicating with the Nomad API. The client part will implement the user interface and the OpenID Connect authentication. List of both the functional and non-functional Bedouin requirements are listed in Table 3.1.

1. Functional requirements
1.1. Read template and template parameters files
1.2. Render templates
1.3. Submit job to Nomad
1.4. Update job by its template
1.5. Stop the job
1.6. Audit information about a user
1.7. Monitor ongoing job
1.8. Read task logs
2. Non-functional requirements
2.1. TypeScript and React
2.2. OpenID Connect authentication
2.3. Dockerized
2.4. Stateless as possible
2.5. Provide logs

Table 3.1: Bedouin requirements summary

Chapter 4

Development technologies and tools

4.1 Development technologies

4.1.1 Frontend

Currently there is no commonly adopted alternative to JavaScript when running a dynamic web application in a web browser. At the same time, when developing a web application, vast number of languages, frameworks or build systems can be used. [16]

4.1.1.1 TypeScript

Even though JavaScript language is evolving fast, the developer community is moving even faster. [17] Today, there is a number JavaScript-based languages, JavaScript supersets or syntax extensions. A possible way to run a code written in non-standard JavaScript in a browser is to transpile it to naked JavaScript.

TypeScript is a superset of JavaScript that primarily focuses on enhancing JavaScript by static typing of code. Static typing leads to less error-prone code and enables intellisense [16] features to be used in IDEs. Moreover, TypeScript provides language features of the future JavaScript by implementing the proposals for the JavaScript language development.

TypeScript allows developers to gradually create the code typings of the existing JavaScript code, without touching it, in a separate module. TypeScript is gaining popularity as well as the number of typed libraries is growing.

[16] Intellisense is a code completion functionality of IDE (integrated development environment).

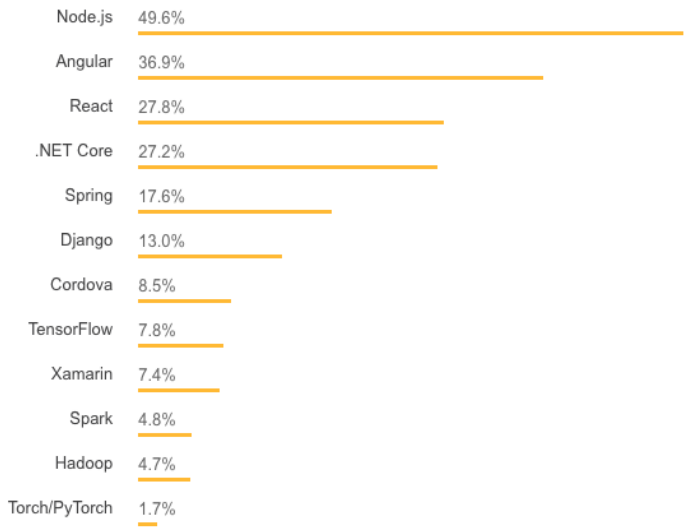

Figure 4.1: Popularity of frameworks, libraries, and tools

Source: StackOverflow survey [15]

4.1.1.2 React

React framework is nowadays one of the most popular JavaScript frontend frameworks. It is even one the most popular framework in general, according to StackOverflow survey 4.1. React is maintained by Facebook and its development is also driven by the developer community.

React is a frontend framework concerned with the user interface. In order to create more complex web applications, using external libraries for state management or API calls is almost inevitable. React itself does not provide as much functionality as the competing frameworks, such as Angular or Vue. But React, dealing only with the UI, gives the developers opportunity to create a project based specifically on the project needs using the dependency libraries of their choice.

React components React component is an elementary building stone of React applications. One React component represents one part of the UI and can be composed of other components. The component can work with the data that are either passed to the component as a props [17] or the data that component holds as its state. The essential part of a React component is a render function. The render function describes how the component and its data should be presented using other components. For rendering React components to the HTML markup to be interpreted by the browser, React comes with ReactDOM library implementing base components conceptually

[17] React props are arguments of a React component

representing the HTML DOM elements. Render function is of the component's lifecycle methods. Other methods are, for example:

- `componentDidMount` (This method gets called when the component instance was inserted in a DOM for the first time. It is a good place to set up subscriptions or initiate API call to an endpoint. Also, this is the place to trigger initialization the needs access to the DOM.),
- `shouldComponentUpdate` (React triggers component's rendering whenever its props or state changes. If it is known when the change won't affect the result of the render function, React can be informed by this method. This approach optimizes application performance.),
- `componentDidUpdate` (If an interaction with the updated DOM after it was re-rendered is necessary, this method is place to do so.)

and others.

Virtual DOM The HTML markup is a declaration of a web page structure that the web browser reads in order to paint the actual page contents for the user. Document Object Model (DOM) is an API used to access and control the HTML. Purpose of the ReactDOM library is to manipulate the DOM, in order to visualize the React components. Operations involving changing the DOM and thus repainting the page are expensive. But every time that React component runs the render function, all of its child components gets returned, not only those that has changed. To abstract the programmer from dealing with the optimal DOM rendering using only the components that has changed, ReactDOM implements a concept known as virtual DOM. Virtual DOM is an in-memory tree of the JavaScript objects, conceptually representing the DOM elements. Virtual DOM handles the redundant renders of the elements that have not changed and performs only the DOM changes of the elements that have changed.

Functional programming React plays well with the idea of functional programming. Developers are encouraged to think of the components as presentational components and container components (or "dumb components" and "smart components").

Presentational components serve the purpose of rendering the actual UI elements. These components do not hold any state. Instead, they only work with the data that has been passed to them from parent components. Presentational component is basically a pure function that receives a state and returns UI.

Container components on the other hand should not render any of the UI elements, they should only render other implemented components. Their purpose is to provide the state to the presentational components and handle

callbacks coming from user actions. The container components are called "high order components", which refers to the high order function paradigm of the functional programming.

The presentational and container component pattern enables components to be more generic and thus reusable. Container components can be composed together, so each of the components can focus on distinct part of the functionality[18].

React context The usual way to pass data through a component tree is by using props, but for some kinds of data this approach can be very inconvenient. Typically, it is the kind of data that needs to be shared across the application. An example can be the data about the user preferences. There may be a case when there are just few components affected by the user preferences. Assuming that the user preferences data is stored at the root of the component tree, then all the components that are part of the component tree to the affected components need to pass the data through.

A solution to this a React context. React context provides a place for storing data allowing the developers to subscribe for it anywhere within the component tree from the point where the context has been declared. There are two kinds of the React high order components to interact with the context. First, the context provider, can be inserted anywhere in a component tree exposing the context to be subscribed anywhere in its descendants. The second component is the context consumer. When it is inserted in the context provider's component subtree, it gets access to the context data.

JSX React comes with a support for a special HTML-like syntax for describing the React components called JSX. JSX expressions makes the code of the component render function look like an ordinary HTML markup, but behind the scenes the actual React components are being created. The following two expressions are equivalent.

```
<div>Hello CVUT!</div>
```

```
React.createElement('div', null, `Hello CVUT!`)
```

4.1.2 Backend

The backend part will be written using TypeScript, as well as the frontend part. This approach is beneficial for as it enables code reuse and requires the maintainers to master single programming language. The server will take care both of serving the frontend code and assets to run in a browser, as well as providing an API for the frontend calls.

4.1.2.1 Node.js

JavaScript was initially created to run in the web browsers as part of a dynamic webpage. Eventaully, as the language got more evolved, Node.js was introduced to enable JavaScript code to run on a server as well. Node.js uses V8 JavaScript engine that is used in Google's Chrome browser and is open-sourced. The JavaScript environment is also provided with access to system functionality such as networking and controlling the file system.

Part of the Node.js ecosystem is Npm (Node Package Manager). It is a command-line tool managing third-party packages used in a JavaScript application. It also provides a way to run scripts, for example to build or to test the application. Part of the Npm project is a public package registry, which is currently by far the biggest one in terms of number of registered packages. [19]

4.1.2.2 Express.js

Express.js is a JavaScript framework for building HTTP APIs. It is a very lightweight framework that handles HTTP requests by pushing it down the chain of Express middlewares.

A middleware is a plain function that takes from 3 to 4 arguments. First, it takes an object representing a request that was received by a server. Next argument is an object representing a response that is being created and eventually is sent to the client. Third argument is a 'next' function that invokes next middleware of the chain. Optionally, the middleware used to handle errors arised while processing the requests. In such case, the middleware function takes a forth argument — an error. A middleware can perform whatever actions with the request and the response, edit or add properties to it and finally can either invoke execution of the next middleware, or end the middleware chain by dispatching the response to the client.

4.2 OpenID Connect authentication

OpenID Connect is an authentication protocol used for user identification using an authentication authority. This protocol allows service providers to use existing authentication functionality of a third-party authority. This approach is very commonly seen in many web services, allowing a user to sign up to the service by their account of another service. For example signing up to Spotify using user's Google account. OpenID Connect is based on the OAuth 2.0 protocol.

4.2.1 OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authentication. It is based on access tokens granting a client access to resources of a given system. OAuth 2.0 defines several grant types, describing different ways of how the access token is obtained. [20]

Authorization code The authorization code grant type is the most commonly used grant type. Its main benefit is that a client is able to get access to resources placed on a resource server. User only needs to interact with the authentication provider, without exposing their credentials to the resource server. The user sends their credentials directly to the authentication authority and they get an authorization code in exchange. This access token is then used to proof the user identity to the resource server. As a last step, the resource server issues the user an access token, which is used in all the following communication from that point. [21]

Token formats The OAuth 2.0 framework can be used with different formats of an access token. There are 2 common ways to implement access tokens: string of hexadecimal characters; or structured tokens, such as JSON web tokens (JWT). [22] The token of the first case is a string without any special meaning, its only purpose is to carry a value that represents a secret. Its authenticity has to proved by the authentication authority. JSON web tokens are on the other hand self-contained, meaning their authenticity can be proved just with knowledge of the public signing key of the issuer. No communication with the authentication server is needed. Furthermore, JWT can contain payload, typically information about the user and its authenticity is granted as well.

4.2.2 OpenID Connect

OpenID Connect exploits the OAuth 2.0 authorization code grant type in order the client to obtain an access token. In the following step, the client sends an additional request to the authentication server, getting an ID token in response. An ID token is JWT formatted token enclosing information about the user in its payload.

The communication flow between interested parties of the OpenID Connect protocol is explained in subsection 5.2.1.

4.3 Build and containerization

4.3.1 Webpack

As mentioned in 4.1.1.1, writing the application in non-standard JavaScript requires transpiling the code. Moreover, is it a good practice for the frontend code to bundle and to minify it, in order to decrease the load time of the page.

Webpack is a modular bundling tool. First, Webpack builds a dependency graph using the entry file(s) to the code. Then, using transformation middlewares (called "loaders"), it loads up all the dependency files. This is where the transpilation to JavaScript happens. Loaders does not have to be used just for transpiling the code, they also can be used for loading and parsing other file contents, such as JSONs, CSS files, images, etc.; for code linting[18]; and for many other uses. Loaders are an arbitrary code, so there is no limitation in what they are able to do. They can also by chained together or enhanced by additional plugins.

For the frontend development, Webpack can be used with a development server, to serve the bundled application code at the time as it is being written. When running Webpack in a "watch mode", it runs the bundling process on every file change and automatically reloads the page in the browser that is running the app.

While automatic building and running the app is very useful and time-saving feature, realoading the whole application and thus losing the application state slows down the development process. In order to avoid full reloading, Webpack comes with a feature called Hot Module Replacement (HMR). On the serer side, when the source code is changed, the HRM plugin sends the changed code, along with a manifest, to the browser via WebSockets. The frontend application runs on top of the HMR runtime, which gets called by the WebSockets to update the chunk of code that has changed. If the update is not possible then a full page reload is required.

One other Webpack feature worth mentioning is "tree shaking". Webpack detects unused parts of code (so-called "dead code") using static code analysis and removes it. While this is useful for eliminating application's own unused code, it is especially useful for shaking off the unused parts of the external libraries.

4.3.2 Docker

Docker containerization is a trending method used to pack and deploy applications. The main advantage of this approach is portability and encapsulation of the application itself, its configuration, and all of its dependencies. Docker is nowadays widely adopted by companies and service providers. [23]

[18] Code linting is a process of analyzing code for potential errors.

Docker introduces isolated environments called containers. Among other things, containers allows developers to pack the application along with the needed resources and to run it inside container's operating system. Docker shares the kernel of the hosting operating system to by used by running containers, allowing the containers to be very small in contrast with virtual machine images.

Chapter 5

Implementation

All the project code is placed in a single repository: the backend, the frontend, and the shared code. Both the frontend and the backend code has it's own build process set up, each pointing at the entry file of each part of the solution.

Bedouin code structure is conceptually broken down into distinct modules (Figure 5.1). These modules forms a specific functionality of a program, but they are tightly coupled, in a way that they do not provide or implement predefined interfaces. Concerning the size of this project, such approach would not be effective.

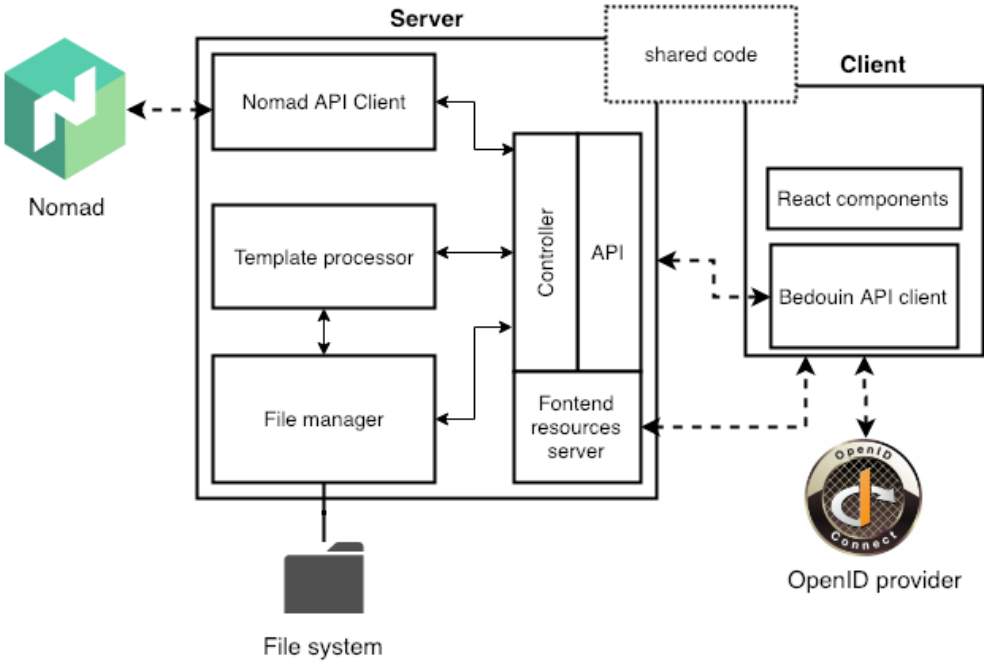

Figure 5.1: Internal implementation structure

```
src ..... base folder for application code
  client ..... frontend code
  server ..... backend code
  shared ..... code used both in frontend and backend
public ..... frontend HTML resources
build ..... webpack build output
tests ..... testing assets
webpack.config.client.js .... webpack build configuration for frontend
webpack.config.server.js .... webpack build configuration for backend
```

5.1 Backend

5.1.1 Template file manager

The storage of template files, both template specification and template parameters specification, is on a file system. This is one of non-functional requirements of this application. Each template file and template parameters file need to be placed in its own folder. All these folders are then placed in a single root folder. The application is aware of the folder location by specifying it in a configuration file.

The Bedouin app implements a template file manager module for interacting with template related files on a file system. Once the app starts, it reads through the root template folder and identifies all the template related files based on a file extension. It then loads up the file contents and calculates checksums of all the template files using the md5 hash algorithm. Finally, it saves the template folders structure, along with the checksums, in the in-memory cache. The cache saves the system resources, so it needn't access the file system on every request. When the cached data expires due to set TTL (time to live) expiration parameter, files get loaded again. So it handles file change automatically, after the specified time limit. When it is needed to load the changed files immediately, the user can force the cache to be deleted and the files get loaded from the file system again, when they are requested.

5.1.2 Template renderer

The Bedouin templating syntax follows the Levant templating syntax in order to provide compatibility between the two applications. Levant is written in the Go programming language, so is Nomad. Levant can thus leverage some of the Nomad features by importing its packages, since Nomad is open-sourced.

Levant uses Nomad features on many places, mainly exploits its API client to communicate with the Nomad server. It also takes advantage of Hashicorp's Terraform library, where the HCL[19] is defined. Levant uses HCL parser to read its template variables specification files that follow HCL syntax.

[19] Hashicorp Configuration Language

The Levant's templating functionality is implemented on top of the template package of the Go language. Levant template variables are of two kinds. The first kind are variables that are interpolated by values provided by the user. The other kind are keyword variables that are interpolated by functions passed into the Go renderer. Levant implements these functions and passes them to Go template renderer.

It would be of great benefit to reuse the Levant's template functionality as well as the HCL utility functions. Not only would it save time to develop Bedouin, but it would be ensured that the functionality behaves exactly the same way. Also, in case of an update of these features in the source projects, it would effect Bedouin's functionality automatically.

Unfortunately, there is no direct way to use code written in Go in a JavaScript application. However, there are two possible solutions to make Go and JavaScript work together: Go to JavaScript compilation and Go to WebAssembly compilation. In the first case, there are two existing compilers possible to use, GopherJS and Joy. In the latter case, Go language from version 1.11 ships with an experimental port to WebAssembly.

5.1.2.1 WebAssembly

WebAssembly addresses the problem of JavaScript being the only supported programming language of web browsers. WebAssembly defines a binary format that has a potential of running on the web at near native speed. On side of that, WebAssembly defines corresponding language with syntax and structure. The language is very similar to assembler and it is not mainly intended to be written by hand. Is it designed to be a performant compilation target for other languages, preferably low-level languages, such as C or C++. [24]

WebAssembly is capable of running and communicating with the JavaScript code. It brings great new possibilities for the web applications to run a piece of software written in other language with very good performance. It allows to deliver complex and computational intensive applications such as games or 3D visualizations to web browsers. WebAssembly aims to be language-, hardware-, and platform-independent. [25]

WebAssembly became a web standard within World Wide Web Consortium (W3C) and is currently supported by all[20] major web browsers. From the Node.js version 8.0.0, WebAssembly is supported as well.

Go compilation to JavaScript

The Levant code compilation to JavaScript was not possible using Joy. The Joy compiler from Go to JavaScript is unable to compile most of existing standard Go libraries [26]. Its development is currently on hold in favour of WebAssembly.

[20] Edge, Firefox, Chrome, Safari, Opera

WebAssembly seemed to be a very promising option, since there is an official support of the Go language project, but the Go to WebAssembly compilation is still an experimental feature and in the case of Levant, it failed due to system-level calls in the Go code.

The compilation using GopherJS was the only successful one. Steps taken to make that happen were:

1. The desired render to function to compile would take arguments template and parameter values; and would return template with interpolated parameters. Unfortunately, Levant render function is actually a method of Levant-specific template class, thus cannot be directly compiled as a standalone function. It was needed to write a new Go function that follows the desired function signature and exploits the internal Levant code. Since it is not possible to access the internal code of a library from outside, a tiny portion of the Levant code had to be modified to allow it.

2. Create a new Go file that imports GopherJS package and the package containing targeted function to be compiled to JavaScript. Then use GopherJS utility function to point the targeted function to be exported as part of JavaScript module export.

3. Use a GopherJS command line tool to compile the Go file from the previous step. The resulted JavaScript file is ready to use.

Since GopherJS does not support tree shaking of a dead code, the resulted JavaScript code was approximately 20 MB. Compiled file contained all the Levant's dependency libraries, both Go libraries as well as third-party libraries. Webpack tree shaking does not help in this case, because it does not handle files that big and errors with a memory overflow. This is an issue that comes with a bit of performance slowdown, but fortunately, since this code is running on a server, the file size is not crucial.

The issue can be solved by distilling the source code to the minimal functionality needed and then compiling it again. This does not fit into the current time scope of this project, but it is an issue to focus on in the further development. It is also possible that some time in the future GopherJS or the Go to WebAssembly compiler will support the tree shaking of code. If this was the case, the source code could stay as it is.

5.1.3 Nomad API client

Nomad provides RESTful HTTP API both for interacting with jobs as well as for monitoring the jobs, allocations, etc. For a one-time request of a Nomad state, ordinary HTTP request can be used. When it is needed to continuously monitor the Nomad state, the API handles that using long-polling technique. And for observing task's stdout and stderr logs Nomad uses HTTP streaming.

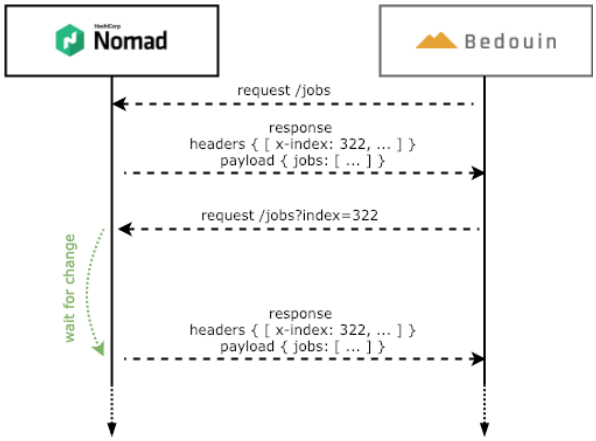

Figure 5.2: Nomad blocking queries

Nomad long-polling

For endpoints that supports long-polling requests Nomad introduces the index parameter that represents Nomad state at a given time. The index holds an integer value and the higher the value is the newer state it represents. This index value is used by a Nomad to distinguish whether the client needs to be updated with newest state. This feature is called "blocking queries". [27]

Blocking queries schema

1. The client sends an ordinary request to a Nomad endpoint.
2. Nomad sends a response with a header field containing the index.
3. The client sends another request, this time with the index value as a query parameter. Optionally also with a timeout parameter.
4. If the state of the endpoint already changed from the time of the previous response, Nomad responds right away with a new index value. If not, Nomad puts the request on hold until the state changes and then responds. In case nothing changes within a timeout interval (either provided by the client or a default value), Nomad sends a response with an index unchanged.
5. Each subsequent request follows the flow from step 3.

Bedouin's Nomad API client module implements a function that follows the blocking queries flow of a provided endpoint. This function uses the Observable pattern for notifying the application whenever the state of the endpoint (the index value) changes.

Currently the API client is part of the Bedouin repository. Once it is fully developed and mature, it will be separated to a standalone library and published to the Npm public registry. In this way, it will be possible to reuse the code in any JavaScript project.

5.1.4 Server

The server part consists of the HTTP server and the API built on top of it using ExpressJS framework. Apart from the API, HTTP server is used to provide the Bedouin frontend application code and other resources to the browser. But this is just the case when Bedouin is used in production. While developing, frontend resources are served by webpack-serve which comes with powerful features such as Hot Module Replacement. Also, it comes with a proxy that redirects all the API requests to the actual backend.

5.1.4.1 API using Express.js

Express.js sends every incoming request through a series of middlewares, as described in subsubsection 4.1.2.2. The Express library provides router middleware interface. Router middlewares can be nested and each one can be used to implement a specific part of the API. This way the code of API endpoints can nicely be broken down into modules and then put together along with routing setup in a single place.

Each API endpoint is broken down to two parts: an API route handler and a controller. The route handler is a part of an endpoint that takes care of interacting with the request and the response objects: checks validity of a request, parses request parameters, writes data to response, handles errors, etc. The second part, the controller, is a set of functions that provides interaction with the rest of the application. This provides a little bit of abstraction for the API route handlers.

Authentication The server part of the Bedouin authentication mechanism is very simple. The server takes part in the last phase of the authentication flow described in subsection 5.2.1.

After the frontend obtains the JWT access token, it sends the token in every subsequent request to the server. The server handles the verification of the token authenticity in one of top-level Express.js middlewares. If the authentication feature of Bedouin is enabled, the authentication-handling middleware makes the following steps:

1. It looks for the corresponding request header containing the JWT access token.

2. It Verifies the JWT signature using the OAuth 2.0 provider's public key. The key is specified in the Bedouin configuration.

3. If the token is genuine, then it parses the user information from the JWT payload and attaches it to the request object, in order to be used by the following middlewares. If the token verification fails, then the middleware ends the execution of the middleware chain and sends the appropriate response to the frontend.

The user information contained in the JWT payload is then used to label the Nomad job with the name of the user who submitted it.

Error handling Bedouin's API code takes advantage of error handling middlewares of the Express framework. When an error occures somewhere in application's code and eventually hits the API route handler, the handler can simply cancel execution of the rest of the handler by calling the error middleware, or chain or error middlewares, and pass the error to it.

The application declares an APIError class, that contains properties such as HTTP error code, reason of an error and optional additional parameters. An API controller's job is handle errors that possibly happen at some point of request execution. If an error is known, then it is translated into an APIError with appropriate parameters, otherwise it is rethrown as it is. The next thing that happens is that an API router handler calls the error middleware. Lastly, the error middleware checks for the error type and if the error is of type APIError, then the error middleware composes a response by the error's properties. If the error is of other type, then the middleware responds with unknown server error status code.

5.2 Frontend

5.2.1 OpenID Connect client

The frontend part of Bedouin handles the part of the authentication flow (Figure 5.3) as it is defined by the OAuth 2.0 and OpenID Connect protocols.

The authorization code grant flow

1. The Bedouin frontend initiates the flow with (a) redirecting the web browser to the (b) OpenID provider login page (Keycloak).

2. After the browser displays the login page, the user inputs their credentials and submits them back to the OpenID provider endpoint.

3. Assuming the credentials are valid, (a) OpenID provider redirects the web browser to (b) the Bedouin frontend using a URL with the authorization code included.

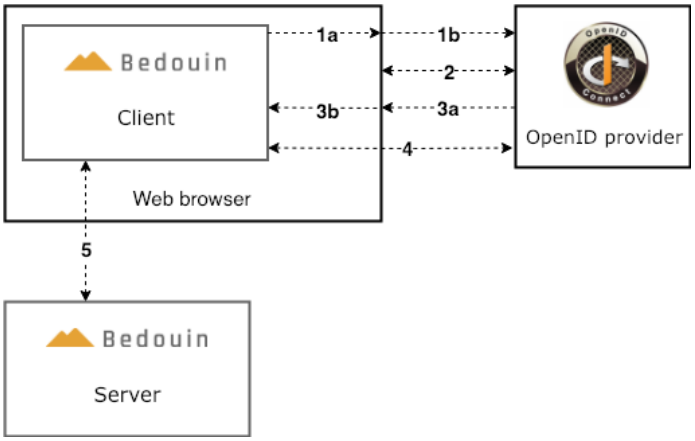

Figure 5.3: Bedouin OpenID Connect flow

4. The Bedouin frontend then sends one more request to the OpenID provider in order to exchange the authorization code for ID token (containing the information about user).

5. Finally, the frontend stores the ID token as a cookie and includes it in all the subsequent requests to the backend. The backend validates the token signature and retrieves the user information from the token's payload.

5.2.2 React UI

Being a good practice to split the React components to so-called "smart" and "dumb" components, the Bedouin app follows this pattern by introducing "API components". "API components" are used to manage the state of a given API resource. They use Bedouin API client functions to retrieve the state of the resources and then save it to component's state. All of this state is then passed down to child components. "API components" are the "smart" components in this case, so the child components do not have to be aware of the API functions. They just need to deal with rendering the UI content for the user.

5.3 Shared code

5.3.1 Typings

Both frontend and backend is implemented using TypeScript — a statically typed superset of JavaScript. Big advantage of a fullstack TypeScript web application is a possibility to share the typings of objects, that are used to transfer the data between frontend and backend.

TypeScript object interfaces declared in Bedouin's Nomad API client are reused within the whole application.

5.3.2 Error codes

Shared code also includes enumeration of error codes of the Bedouin backend. When the backend encounters an error and needs to inform the frontend of the problem, it sends a response to the frontend with an error code. The frontend then picks up the code and takes any actions needed, according to the error code. To ensure consistency of the error code throughout both the backend and the frontend, it is placed in a shared code.

Chapter 6

Testing and production

6.1 Testing

Software testing is an important part of the software development lifecycle. Its purpose is to ensure that the software product or its components satisfy given requirements and properties. Software testing is a wide subject comprising many different testing levels and strategies, focusing on a certain dimension or certain quality of the software product. Creating a testing suite for a certain software product is highly dependent on circumstances: project size, development team size, software complexity, criticalness, etc.

While developing software, it is good practice to have a set of tests in place to run whenever there is new code pushed into the source code repository. This is very important when working in a team of more people and especially in case of a loosely tied team such as the open-source community. Testing the changes made in code should prevent programmers from breaking the existing functionality by accident. This is called regression testing.

Creating a testing suite covering every possible failure scenario in complex software applications, such as web application, can result in very complex and time-consuming testing solutions. In a real-world application, it is often a trade-off between complexity and maintainability of tests and a risk of an error.

The Bedouin application implements tests of the Bedouin backend API. The tests servers the purpose of regression testing.

6.1.1 Bedouin testing tools

Postman Postman is an API development framework suitable both for designing an API using API mocks and for testing the API.

Postman comes with a powerful user interface allowing developers to very quickly and easily query API endpoints. It is built on top of Node.js runtime allowing to use JavaScript for creating scripts to be used in the Postman

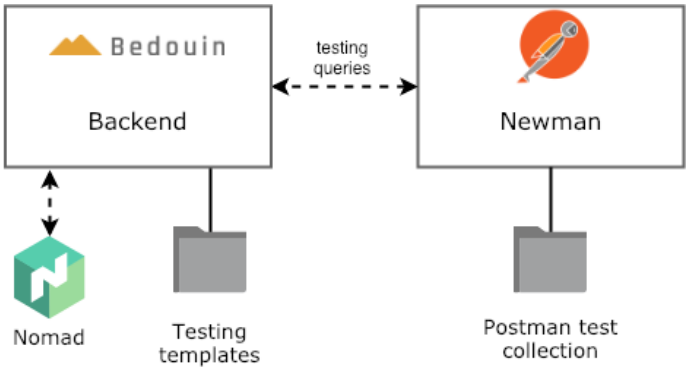

Figure 6.1: Bedouin Newman testing schema

environment. For example to run complex validation logic on a response that came from a server, or to pass data around between multiple requests.

Newman Postman can be integrated with a command-line tool called Newman, which is able to run test collections using created using Postman. Newman maintains the feature parity with Postman. [28]

Postman serves very well the purpose of ad-hoc requesting the API while development. This approach allows developers to create API tests by reusing these requests, make them more generic using parameters, enrich them by pre-request and test scripts, wrap them in workflows and finally export them to be run by Newman.

6.1.2 Bedouin tests

Bedouin tests are created using the Postman-Newman tandem described above, and they are being controlled by a shell script. The testing assets include template files that are used by the Bedouin backend while testing it. The testing process assumes there is a local Nomad instance running on a machine.

Testing script Before the Newman tests are executed, it is needed to configure and start the Bedouin backend. That is the purpose of the shell script. The script steps includes:

1. Build the Bedouin backend
2. Set the environment variable, that specifies the template files location, to point at the testing template files.
3. Run the Bedouin backend
4. Wait for the first backend health-check indicating it is up and running

5. Execute Newman tests
6. Stop the backend

6.2 Ataccama use case

The first use case to try the Bedouin application in Ataccama is deploying an instance of Ataccama ONE. Ataccama ONE is a platform for data management including for example: data discovery, data profiling, business glossary, data quality management, and other features.

The Ataccama ONE platform consists of several modules, that can also perform as standalone applications, having its own APIs. These modules are connected and bundled together, with a web application as a gateway for users. As a result, Ataccama ONE is built as one large monolithic backend with two side modules and a web application frontend.

6.2.1 The job template of Ataccama ONE

The Nomad job used for Ataccama ONE deployment consists of six tasks grouped into two task groups. One for the frontend part and one for the backend.

6.2.1.1 Backend task group

The backend task group holds three tasks. First is the monolithic backend described above. The other two are external modules connected to the backend local network within the allocation.

6.2.1.2 Frontend task group

The frontend task group contains three tasks: the application, a maintenance page, and Nginx[21]. Each one is running in a Docker container.

Application First is the frontend application itself. All the assets of the frontend application that a web browser has to provided are mapped to a file system of the allocated node, using the Docker mounting mechanism.

Maintenance page This is a very lightweight task and is deployed much faster than the rest of Ataccama ONE solution. It provides the web server with a simple webpage informing the user that the system is being prepared. The webpage contents are also mapped to the file system.

Nginx Nginx is a web server proxying the monolithic backend API and serving the assets of either the maintenance page or the frontend application, when is ready.

[21] Nginx is a web server, load balancer and reverse proxy.

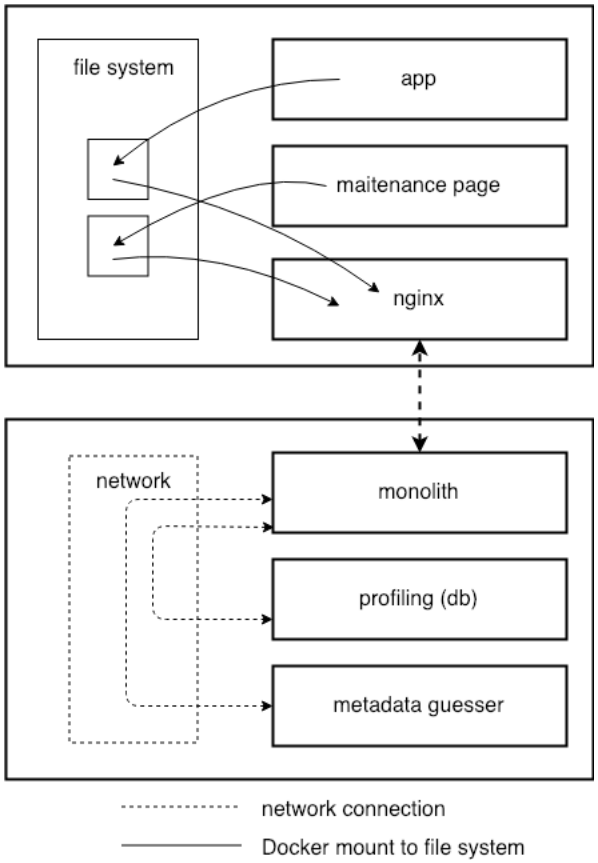

Figure 6.2: The Ataccama ONE Nomad job illustration

6.2.1.3 Template parameters

The template specifies over 10 parameters, including for example:

`FE_APP_IMAGE` ONE frontend docker image identifier

`BE_APP_IMAGE` ONE backend docker image identifier

`NGINX_IMAGE` Nginx docker image identifier

`APP_DEBUG` ONE backend debug settings

`HOST` Host address of the application (for loadbalancer settings)

6.2.2 Running the job

Ataccama ONE has many different configuration files that can modify its appearance and behavior. The consultants are the ones responsible for setting them up, they have a good knowledge of the configuration and are welcomed to

discuss it with the developers, when they run into trouble. The configuration process usually takes some time as the consultants need to continuously test if the changes they have made are effective.

Bedouin was able to successfully handle the template files and submit the rendered job to Nomad. At the time of writing this document, the Bedouin user interface is very simple and it is not fully resistant to user errors, but the core functionality is in place.

6.3 Publishing as open-source

In order to publish an open-source project, it is needed to provide it with a license specifying the terms of usage. A code repository with no license specified is treated as under exclusive copyright, meaning no one can copy or modify the code.

Bedouin is intended to be published without any restrictions, free for both private and commercial use, distribution and modification in any way. Because all of Bedouin's dependency packages are published under "permissive" license, it is possible to use any license. Most common permissive licenses are: MIT, Apache 2.0, ISC, and BSD. [29]

Bedouin is published under MIT license. It is a brief and clear license, satisfying the desired properties described above. This license is most common in Npm packages.

Conclusion

The goal of this work was to design and implement an open-source web application as an alternative user interface on top of HashiCorp Nomad. This application was created at the initiative of the Ataccama company, to be used by its consultants. The application should serve the purpose of reducing manual cooperation of the consultants and IT operators. Requirements for the application was declared by the lead DevOps engineer of Ataccama.

The application was successfully developed and proved by running an existing Nomad job that is used in Ataccama. The application goes by name **Bedouin**. The Bedouin design is described in this document and the source code is publicly shared on GitHub[22] under the MIT license.

The application implements all the core functionality in order to deploy and monitor Nomad jobs using templates. On the other hand, the source code contains many solutions that are not generic enough and many known bugs. At this moment, the source code is not mature enough, making it very hard to other developers of the open-source community to join the project easily. If the Ataccama company participates in further development of Bedouin, then hopefully the project will reach the point of stable release.

As a side product, this paper contains a basic introductory explanation of HashiCorp Nomad, that can be useful to understand Nomad without having a DevOps background knowledge. For me personally, such explanation would be helpful at the time when I started working on this project.

As an author, I would also like to emphasize the importance of the software requirements gathering and continual discussion over the design of the software. While developing this product there was one misconception that led to implementing a functionality was not needed and eventually was removed.

[22] https://github.com/Peracek/Bedouin

Bibliography

[1] HashiCorp. DevOps defined [online]. [Cited 12.11.2018]. Available from: https://www.hashicorp.com/devops-defined

[2] Dadgar, A. Keynote: Nomad use cases [YouTube]. [Cited 1.11.2018]. Available from: https://www.youtube.com/watch?v=D5IVQvbPzcA

[3] HashiCorp. Introduction to Vagrant [online]. [Cited 12.11.2018]. Available from: https://www.vagrantup.com/intro/index.html

[4] HashiCorp. Introduction to Terraform [online]. [Cited 12.11.2018]. Available from: https://www.terraform.io/intro/index.html

[5] HashiCorp. Vault landing page [online]. [Cited 12.11.2018]. Available from: https://www.hashicorp.com/products/vault/

[6] HashiCorp. Consul introduction [online]. [Cited 12.11.2018]. Available from: https://www.consul.io/intro/index.html

[7] HashiCorp. Nomad Schedulers documentation [online]. [Cited 1.11.2018]. Available from: https://www.nomadproject.io/docs/schedulers.html

[8] HashiCorp. Rolling Upgrades documentation [online]. [Cited 12.11.2018]. Available from: https://www.nomadproject.io/guides/operating-a-job/update-strategies/rolling-upgrades.html

[9] HashiCorp. Easy and Flexible Application Deployment with HashiCorp Nomad [YouTube]. [Cited 1.11.2018]. Available from: https://www.youtube.com/watch?v=A6CuZUoINX0

[10] HashiCorp. Scheduling documentation [online]. [Cited 12.11.2018]. Available from: https://www.nomadproject.io/docs/internals/scheduling.html

[11] Phillips, A. The Continuous Delivery Pipeline — What it is and Why itâĂŹs so Important in Developing Software [online]. [Cited 1.11.2018]. Available from: https://devops.com/continuous-delivery-pipeline/

[12] James Rasell, A. H. Levant Templates [GitHub]. [commit 622d734]. Available from: https://github.com/jrasell/levant/blob/master/docs/templates.md

[13] Rasell, J. Levant [GitHub]. [commit 3e65564]. Available from: https://github.com/jrasell/levant

[14] Rosner, F. Cluster Broccoli [GitHub]. [commit d0067e7]. Available from: https://github.com/FRosner/cluster-broccoli

[15] StackOverflow. Developer Survey Results 2018 [online]. [Cited 5.1.2019]. Available from: https://insights.stackoverflow.com/survey/2018/

[16] Kharchenko, N. 8 JavaScript Alternatives for Web Developers to Consider [online]. 2018, [Cited 12.11.2018]. Available from: https://codeburst.io/8-javascript-alternatives-for-web-developers-to-consider-22f8d38bdfa9

[17] Harter, M. Making the most of JavaScriptâĂŹs "future" today with Babel [online]. 2015, [Cited 12.11.2018]. Available from: https://strongloop.com/strongblog/javascript-babel-future/

[18] Banks, A.; Porcello, E. *Learning React: functional web development with React and Redux*. OReilly, 2017.

[19] Module counts [online]. [Cited 5.1.2019]. Available from: http://www.modulecounts.com/

[20] OAuth 2.0 [online]. [Cited 5.12.2018]. Available from: https://oauth.net/2/

[21] Bilbie, A. A Guide To OAuth 2.0 Grants [online]. [Cited 5.12.2018]. Available from: https://alexbilbie.com/guide-to-oauth-2-grants/

[22] Bearer tokens [online]. [Cited 5.12.2018]. Available from: https://oauth.net/2/bearer-tokens/

[23] Datadog. 8 Surprising Facts About Real Docker [online]. 2018, [Cited 9.12.2018]. Available from: https://www.datadoghq.com/docker-adoption/

[24] Mozilla. WebAssembly [online]. [Cited 1.1.2019]. Available from: https://developer.mozilla.org/en-US/docs/WebAssembly

[25] Haas, A.; Rossberg, A.; et al. Bringing the web up to speed with WebAssembly. In *ACM SIGPLAN Notices*, volume 52, ACM, 2017, pp. 185–200.

[26] Mueller, M. Introducing Joy [online]. [Cited 30.12.2018]. Available from: `https://mat.tm/joy`

[27] HashiCorp. Nomad HTTP API documentation [online]. [Cited 15.12.2018]. Available from: `https://www.nomadproject.io/api/index.html`

[28] Postman. Command line integration with Newman. [Cited 8.1.2019]. Available from: `https://learning.getpostman.com/docs/postman/collection_runs/command_line_integration_with_newman/`

[29] Guides, O. S. The Legal Side of Open Source. [Cited 8.1.2019]. Available from: `https://opensource.guide/legal/`

Appendix A

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of the thesis source codes
  - `bedouin` ..... Bedouin implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format