

Vernal

Team 25 Design Document

Winston Ngo, Dennis Pham, Hpung San Awng, Aaron Ni

Table of Contents

Table of Contents	2
Purpose	3
Functional Requirements	3
Non-Functional Requirements	5
Design Outline	7
Service Interactions Overview	9
Services	10
Design Issues	11
Functional Issues	11
Non-Functional Issues	12
Design Details	14
Class Diagram	14
Use Case Diagram	16
Sequence Diagrams	17
Sequence Diagram for User Sign-In	17
Sequence Diagram for User App Creation	18
Activity Flow Map	19
UI Mockups	20
Deployed App Overview	20
Deploying App View	21
Template Catalog Page	22
System Architecture Design Diagram	23
Sequence Diagram of Vernal Torii	25

Purpose

One of the most time-consuming tasks many organizations and developer teams face is having to deploy, scale, and manage applications in the cloud. Many teams find themselves having to sink a considerable amount of time into handling deployments and building the pipelines necessary to get their app into the cloud.

These days, with the software development process becoming increasingly more complicated and with new tech stacks and technologies releasing regularly, it is more important than ever for developers to be able to focus on the development of their product rather than having to set up complicated pipelines just to deploy their apps. As a Platform as a Service (PaaS), Vernal is designed with developers in mind, allowing them to seamlessly deploy their app from their own repository in just a few clicks.

Functional Requirements

** When referring to developer and organizations in this section, we are referring to our users who are developers and teams of developers*

1. User Authentication and Management

As a developer:

- I would like to be able to register for a Vernal account.
- I would like to be able to login to my Vernal account.
- I would like to reset my password if I forget it.
- I would like to be able to edit my profile to add a picture and description. (if time allows).

2. Deployment and Infrastructure Management

As a developer:

- I would like to easily deploy my application from my repository to the cloud.
- I would like the ability to automatically scale my applications for more demand.
- I would like to be able to manage deployments and configurations.
- I would like to be able to remove my application from Vernal.
- I would like to be able to start an app from a Next.js/FastAPI, Next.js/Express, Next.js/ApolloGraphQL, Next.js/TypeGraphQL template.
- I would like to use selection boxes to choose app templates.
- I would like to be able to search for a specific template.
- I would like to deploy a PostgreSQL database and connect it to my applications.
- I would like to deploy a Redis instance and connect it to my applications.
- I would like to view the logs from my deployed applications.
- I would like to be able to programmatically deploy applications to Vernal.
- I would like to be able to import an existing app to deploy (if time allows).

3. Monitoring and Insights

As a developer:

- I would like my application's health to be monitored and for the deployment to be restarted if errors occur.
- I would like insights from Lighthouse audits to improve performance, accessibility and SEO.
- I would like to visualize CPU usage, memory usage, and network traffic for my app.
- I would like to see the CI/CD logs and errors from my app.
- I would like to be able to view my application's live topology in the Kubernetes cluster.

4. Application Management and Customization

As a developer:

- I would like to provide environment variables for my hosted applications.
- I would like to see documentation for each app.
- I would like to be able to search for a specific app.
- I would like to be able to view and edit Vernal application manifests.
- I would like to be able to favorite my most used application templates.
- I would like to enable OIDC authentication for my application to secure user access.
- I would like to run automated SAST scans against my application's source code to find security vulnerabilities.
- I would like to run automated dependency scanning to check for dependencies with vulnerabilities and possible upgrades to resolve them.
- I would like to use a cohesive dashboard for my deployment and scaling needs all in one place.
- I would like to be able to navigate the app using side menus and tabs.
- I would like to receive alerts whenever a new vulnerability is detected, so that I can take action as quickly as possible (if time allows).

5. Collaboration

As an organization:

- We would like our developers to be able to see all of our apps so that they can collaborate on the projects.
- We would like our developers to be able to document their code on dedicated documentation sites within the platform.
- We would like to centrally manage our dev team's cloud deployments through Vernal to ensure consistency and efficiency in our deployment process.
- We would like our teams to be able to have a shared knowledge hub so our developers can expand their skills (if time allows).

Non-Functional Requirements

Architecture

There are 5 main parts of our application architecture: the frontend web UI, backend REST API, PostgreSQL database, Redis, and the Kubernetes cluster(s) to power the entire platform.

The backend API will be written in TypeScript using Backstage, with the underlying web framework being Express. Knex is the SQL query builder used for the database and Keyv is used for access to Redis for caching.

The frontend will also be written in TypeScript using Backstage, with the underlying web framework being React. It will send requests to our API to fetch and update data, as well as execute actions from the user.

The Kubernetes cluster(s) will be the key to powering this entire application.

- Argo CD is used to declaratively manage Kubernetes resources
- Crossplane is used to declaratively manage cloud provider resources
- Istio is a service mesh that enables us to secure, control, and manage network traffic between services
- Kiali provides graph visualizations for operators to monitor and observe network traffic
- Loki collects, aggregates, and stores logs from applications
- Prometheus, Node Exporter, and Kube State Metrics collect a multitude of different metrics about the clusters and applications/services hosted on them
- Grafana is an interactive visualization dashboard that provides alerts, charts, and graphs for collected metrics
- Sealed Secrets is a tool to encrypt secret values and store them declaratively in a git repository (goes hand in hand with Argo CD)

Vernal Operator

The Vernal Operator is a Kubernetes Operator which will provide an API through a Custom Resource Definition (CRD) for developers to deploy applications to our platform. This provides an abstraction over the otherwise thousands of lines of YAML we would have to configure, and is the underlying resource that will be managing the lifecycle of deployed applications.

Additionally, this would allow developers to programmatically deploy and update Vernal applications via an API endpoint through Backstage. This means that developers would also be able to declaratively manage and automate Vernal deployments with GitOps methodologies.

Security

As with all applications, security is crucial to the success of the Vernal. Not only will it be deploying and hosting production applications for our users, but it will also have access to the git repositories where their source code is hosted. On the application side, Backstage already has many built-in features to prevent common security vulnerabilities, such as CSRF prevention. Additionally, we will have RBAC for resources in the application and user authentication implemented via OIDC with Keycloak.

For our one-click authentication solution, Torii, we will have CSRF prevention built-in for hosted applications so that developers do not have to worry about implementing it in their backend code, allowing them to focus on their application's core business logic.

Usability

Developers should spend more time working on their apps and less time on configuring them. With this in mind, Vernal should be very simple to use so developers can quickly navigate the UI to easily build and deploy a project. With simple dropdowns, text fields, and selection menus, setting up projects should be very simple for both new and seasoned developers. Documentation will also be standardized through Vernal through documentation templates for developers to complete with every app.

Deployment

As Vernal is a cloud-agnostic solution, it is deployable to any cloud platform with Kubernetes support. However, we plan to host our instance on AKS.

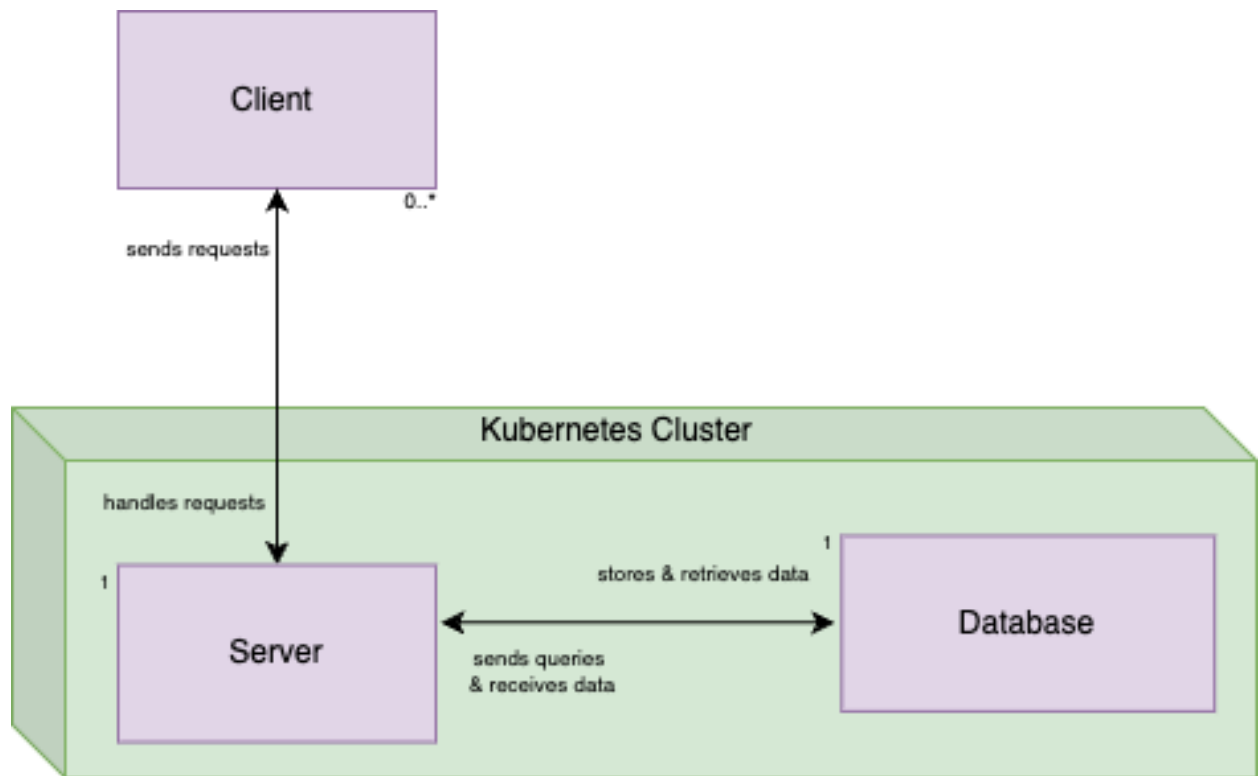
Performance

- Applications deployed on the Vernal platform should be able to handle at least 100 requests per second and more in the cloud since it should scale up automatically.
- When a developer requests to deploy a new application, it should be deployed within 5 minutes.
- When a developer updates the source code of their application, a new, updated deployment should be created within 5 minutes.

Design Outline

Our project, Vernal, is designed as a comprehensive platform as a service (PaaS) utilizing a client-server model to enable developers and organizations to seamlessly deploy, scale and manage their apps. User apps will be deployed on a Kubernetes cluster and can be deployed with other services, such as a database, to ensure a seamless and scalable platform.

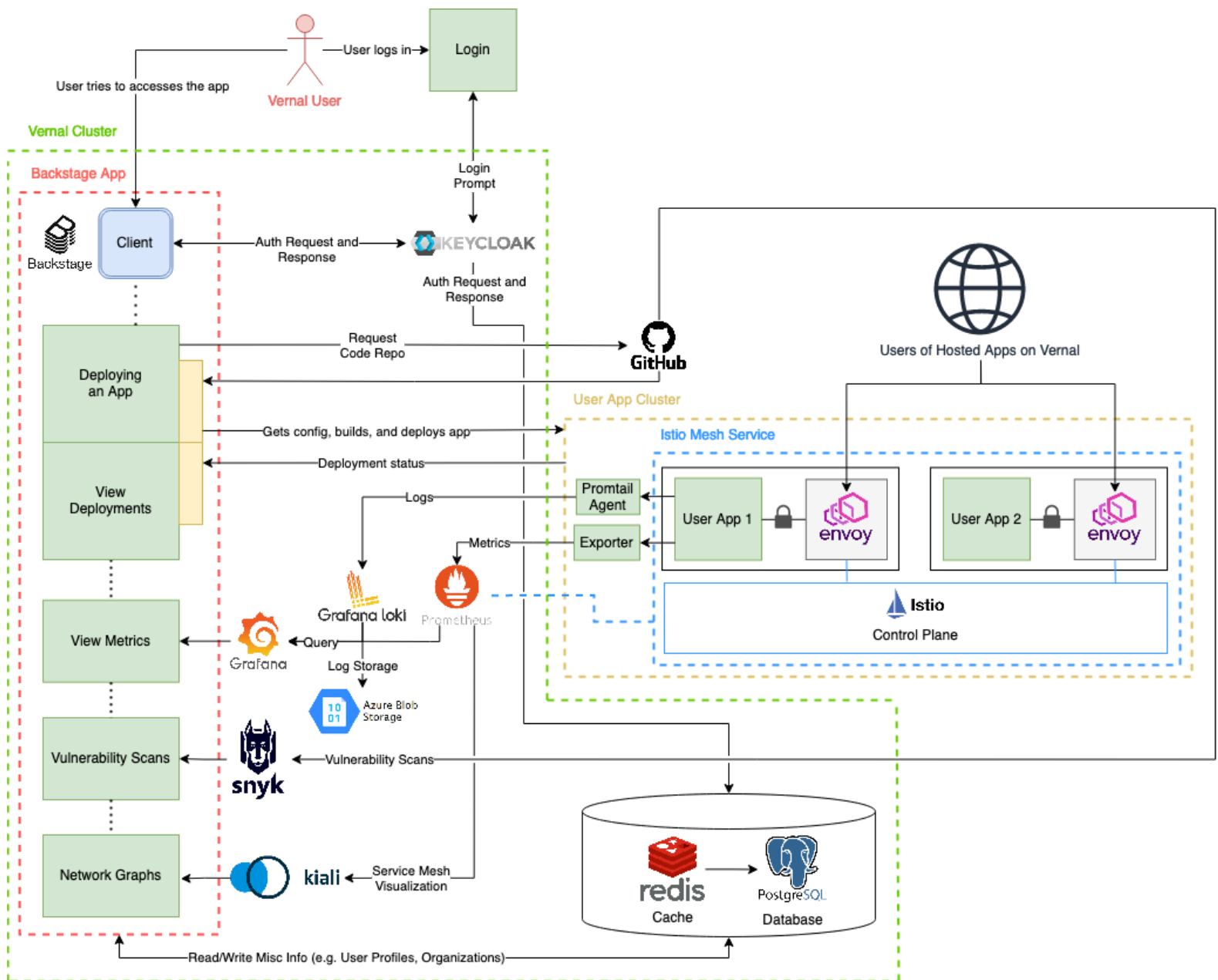
1. Client
 - a. Designed with Backstage for a consistent and modular architecture.
 - b. Serves as the primary interface for users
 - c. Provides comprehensive UI elements to deploy, manage, and scale applications
 - d. Supports performance and health monitoring along with security scanning
 - e. Send requests to APIs to fetch and update data
 - f. Execute actions from the user
 - g. Technologies: Typescript, React (w/ Backstage)
2. Server
 - a. Utilizes TypeScript and Express within Backstage
 - b. Handles application logic and user requests
 - c. Handles interaction with database and Kubernetes cluster
 - d. Fetches and transmits configuration data to client
 - e. Technologies: Typescript, Express (w/ Backstage)
3. Kubernetes Cluster
 - a. Host the deployed applications and databases
 - b. Provide infrastructure for scaling and managing applications
4. Database
 - a. Provides reliable and secure data storage
 - b. Supported by Redis for data caching
 - c. Technologies: PostgreSQL database



Service Interactions Overview

Vernal will utilize a microservice architecture that will utilize a number of independent services that will all work together through service API calls. If something is needed from a microservice, the HTTP protocol is used to make a request to that service's API (POST, GET, PUT, DELETE). All services will be deployed on a Kubernetes cluster (Vernal Cluster) hosted on Azure. When a user deploys their app through Vernal, their application will be hosted on a separate cluster for user deployed apps (User App Cluster). For our database, we will host PostgreSQL on the Vernal Cluster along with Redis as a caching layer for commonly accessed data.

**Second User App Cluster is dependent on the amount of resources we can have*



Services

- Keycloak will be used to handle single-sign on and user authentication.
- Prometheus will be used to monitor apps and handle their metrics.
- Loki will be used to collect and manage any logs.
- Grafana will be used to display metrics from Prometheus and logs from Loki.
- Envoy proxies will route traffic to the correct user application and perform authentication if enabled by the developer.
- Istio will manage all of the Envoy proxies and will create a service mesh with them.
- Kiali will provide visualizations of the Istio service mesh, show what services are encountering errors, and provide data service response times.
- PostgreSQL will be the primary database.
- Redis will act as a cache for handling frequently retrieved data.

Design Issues

Functional Issues

1. What information is required when users are signing up?
 - Option 1: Username
 - Option 2: Password
 - Option 3: Email
 - Option 4: Organization name
 - Option 5: GitHub account

Choice: Options 1, 2, 3, and optionally 4

When creating an account, it is crucial for users to at least provide an email and password to sign up. We will also require users to provide some sort of username as a display name so that they can be identified by other members of their organization. Users can also optionally include an organization name so that they can be added to their organization group. However, users can also choose to do this later on. When deploying an app, we will require users to connect their GitHub account so that they can add their repository to Vernal, but this is not crucial for creating an account.

2. What tech stacks should we support out of the box?
 - Option 1: Modern, popular tech stacks
 - Option 2: Older, more legacy tech stacks

Choice: Option 1

When deciding on tech stacks to support, we will focus on more modern and popular stacks. Although some organizations may still be using older and legacy tech stacks, it is more important for us to focus on supporting modern technologies and provide templates for the most popular tech stacks as they are more commonly used.

3. Should users be able to manually change configuration files rather than just select configurations from a UI?
 - Option 1: Yes
 - Option 2: No

Choice: Option 1

Although the set options that are available in the UI will be enough for most users, some users may want access to more advanced configurations for more niche purposes. If a user needs to modify a very specific configuration, having an in-built editor will let users modify anything in the configuration they need.

Non-Functional Issues

1. What framework will we use to develop Vernal?

- Option 1: Backstage
- Option 2: React JS
- Option 3: Django
- Option 4: Angular

Choice: Backstage

We ultimately decided to use Backstage as it comes with pre-existing tools for building developer portals and a pre-made skeleton dashboard that can be expanded on. Backstage also comes with an expansive plugin ecosystem that can be quickly integrated into any Backstage app. Backstage is also built on top of React, meaning we can take advantage of all the benefits React provides. Since Backstage is built on React, it reduces the learning curve for our team, allows us to use existing React-related resources, and makes expanding and developing our own custom plugins quicker.

2. What will we use for authentication and authorization?

- Option 1: Keycloak
- Option 3: Gluu
- Option 4: OpenIAM

Choice: Keycloak

We decided to use Keycloak because of its open-source nature and its maturity. Keycloak provides a large enterprise-ready feature set that is extremely flexible and widely compatible with standard protocols. Keycloak provides single-sign on and identity brokering through OIDC, OAuth, and SAML.

3. What cloud platform will we use to host Vernal?

- Option 1: AWS
- Option 2: GCP
- Option 3: Azure

Choice: Azure

For our cloud platform, we needed something that provides managed Kubernetes clusters and allows us to provision and manage cloud services programmatically. All of the big three cloud providers give this option. However, our team is most familiar with Azure, so we decided to use Azure over the other two platforms.

4. What database do we use?

- Option 1: PostgreSQL

- Option 2: MySQL
- Option 3: MongoDB

Choice: PostgreSQL

For our purposes, we needed to use a relational database, so databases like MongoDB were already out of the picture. Ultimately, we decided to use PostgreSQL because of its high performance and its support for more advanced features. PostgreSQL is also very reliable and well supported.

5. Will we use monolithic architecture or a microservices architecture?
 - Option 1: Monolithic
 - Option 2: Microservices

Choice: Microservices

We decided to use microservices architecture for Vernal. We felt it was important to be able to quickly and easily scale if needed. We can scale individual services when the need arises, and we have greater fault-tolerance for each service. The microservices architecture means that we have greater freedom and flexibility to choose the technologies we use.

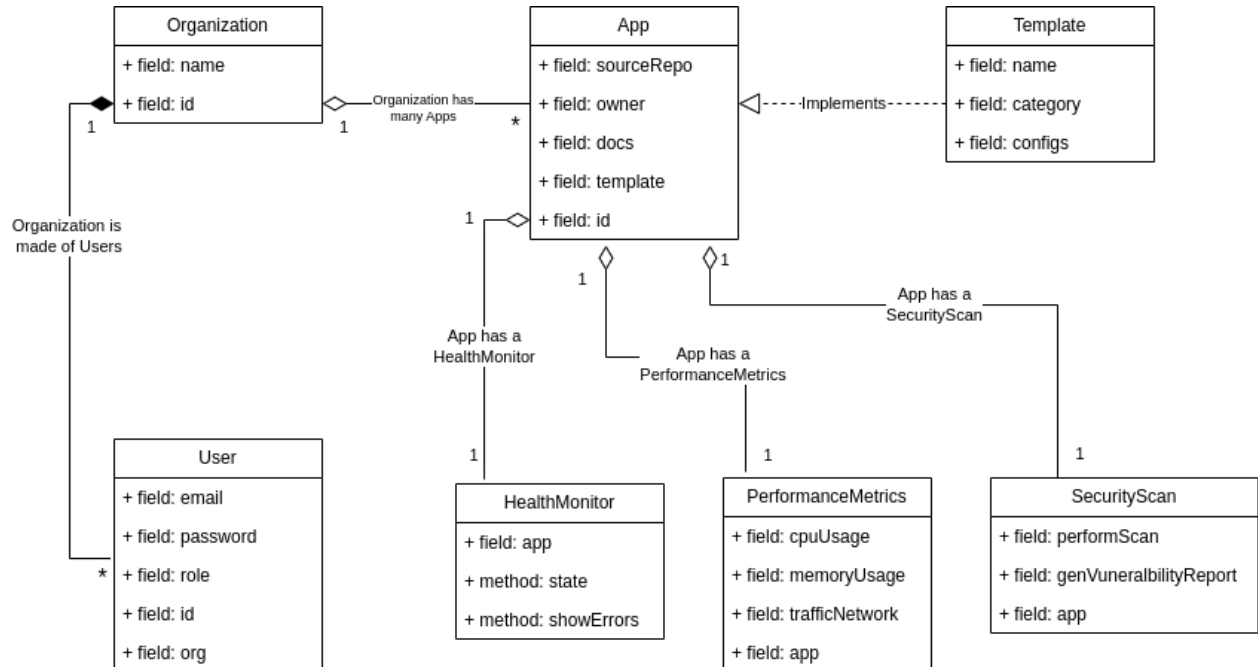
6. How should we deploy user's applications?
 - Option 1: Deploy in containers managed by Kubernetes
 - Option 2: Deploy each application in an individual VM

Choice: Option 1

User applications will be deployed in containers and will be orchestrated using Kubernetes. We decided to utilize containerization because there is less overhead and has better performance. Containerization will let us manage user's apps much faster.

Design Details

Class Diagram

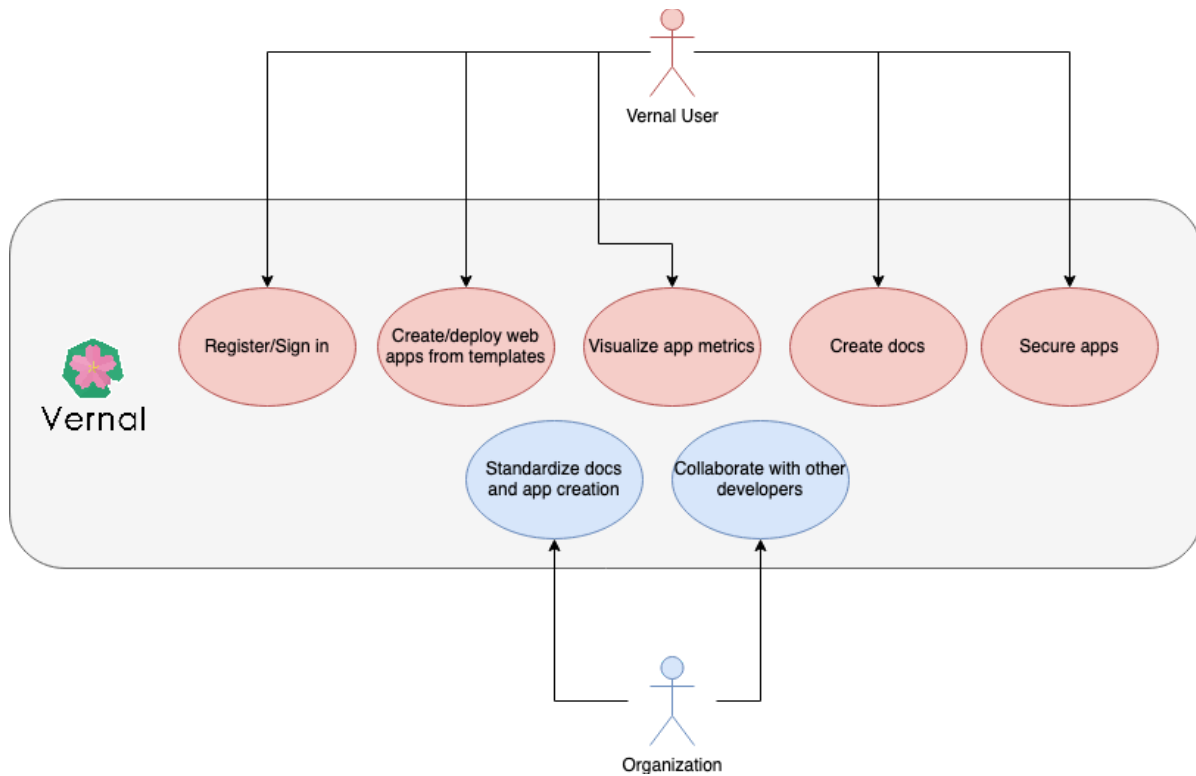


- **Organization**
 - All Organizations will have any number of Users
 - An Organization owns the Apps deployed
 - An Organization can own any number of Apps
 - Users not the Organization cannot see its Apps
- **Users**
 - All Users belong to one Organization
 - Users can have different roles and permissions within an Organization
 - Users can create Apps
- **App**
 - Every App will be derived from a Template
 - Every App will belong to an Organization
 - Every App will have a source code repository
- **Template**
 - There can be any number of Templates
 - Every Template will belong to some category
 - Every Template will have a specific configuration that the App will use
- **HealthMonitor**

- Each HealthMonitor will belong to one App
 - The HealthMonitor will give information about the status of the App, whether or not it is running, and any errors that may have occurred in the app
- PerformanceMetrics
 - Each PerformanceMetrics will belong to one App
 - The PerformanceMetrics will give information related to how well the App is running and the resources the App is using
- SecurityScan
 - Each SecurityScan will belong to one App
 - The SecurityScan will be able to generate a report on any vulnerabilities detected in the App

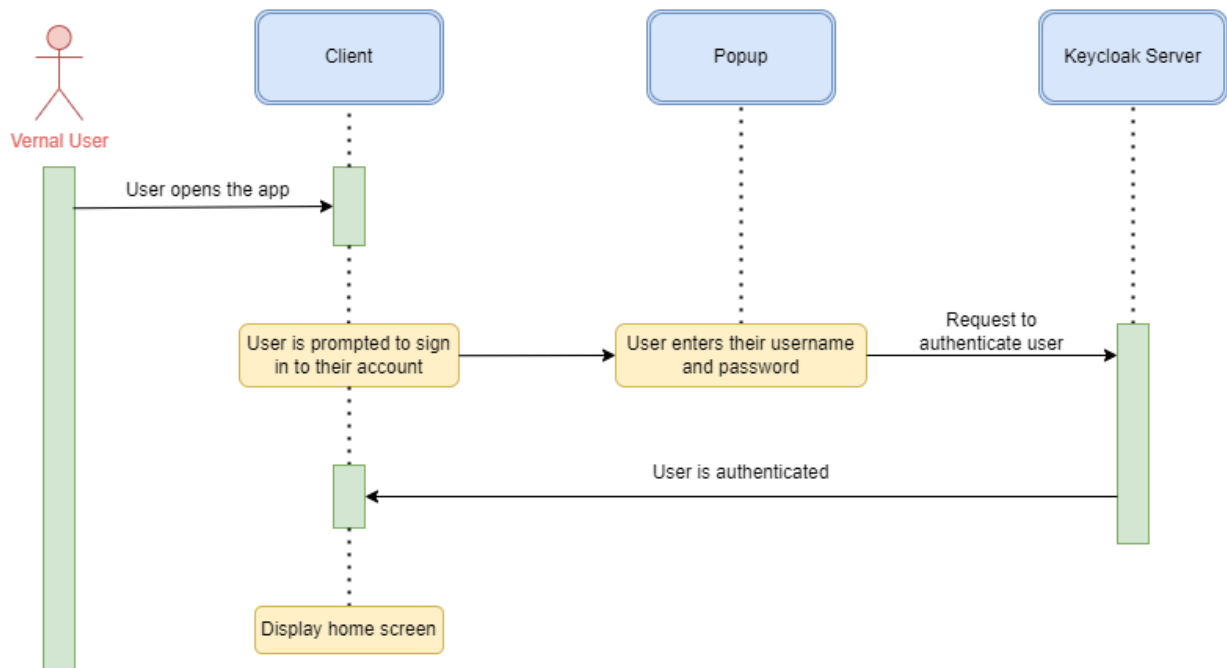
Use Case Diagram

Vernal has several broad use cases for its users and organizations. Organizations will use Vernal to standardize app creation and documentation. Individual users of Vernal will be able to register and sign in, create web apps from templates, visualize their app metrics, secure their apps, and create documentation.



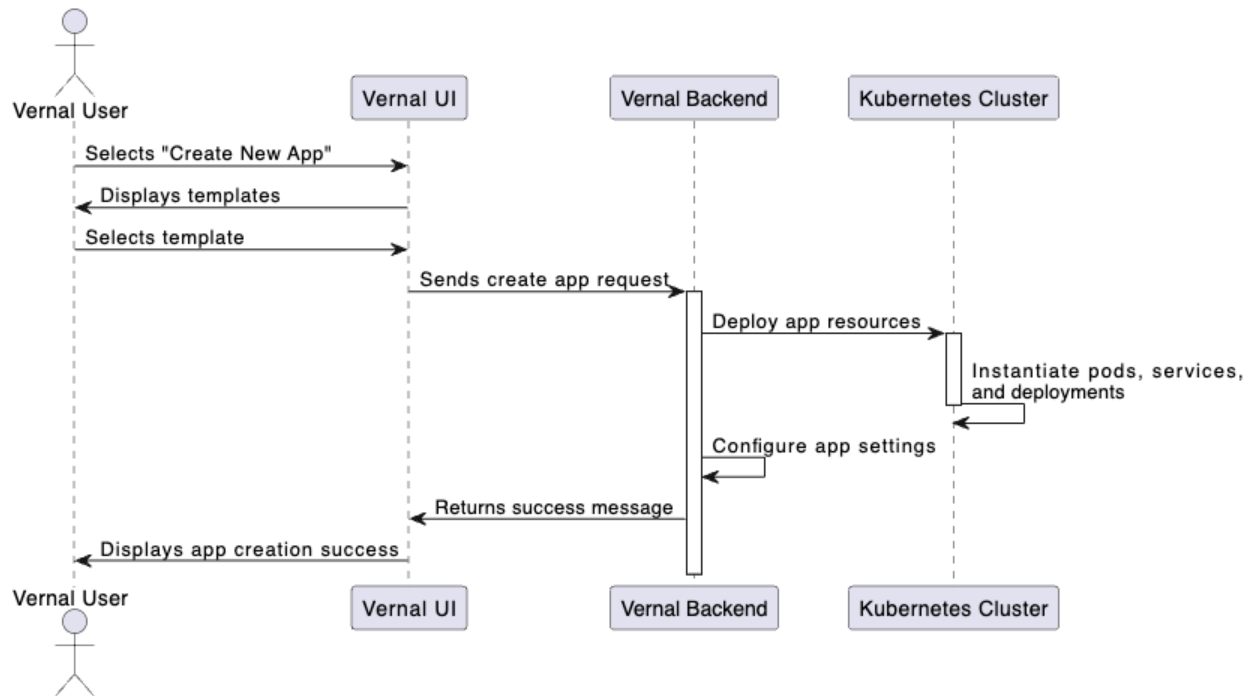
Sequence Diagrams

Sequence Diagram for User Sign-In



When a user opens Vernal, they will be prompted to sign into their account, they will then enter their username and password into a popup. A request to a Keycloak server will authenticate the user and sign them into Vernal.

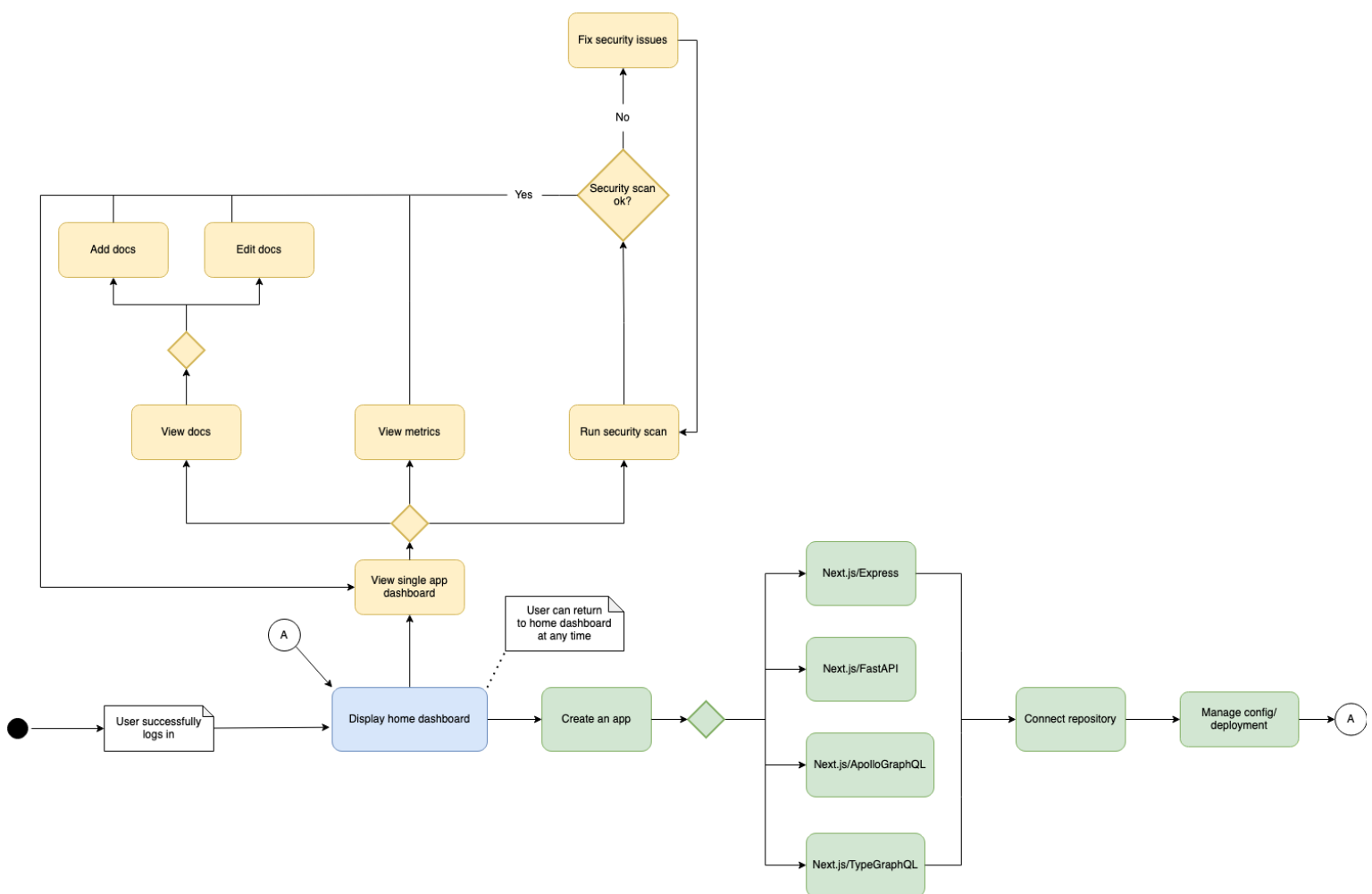
Sequence Diagram for User App Creation



When a developer initiates the creation of a new application in Vernal, they first select the “Create a New App” option within the Vernal UI. A list of templates will be displayed and they can choose which ones they want to create. This submission will trigger a request from the UI to the backend which will then be sent to the Kubernetes cluster to deploy the required resources for the app.


Activity Flow Map

After a user registers and signs into their Vernal account, they have three main ways to interact with the app. A user may create an app, view a single app dashboard, or view all the metrics of an app. When a user creates an app, they will have a choice of four web app templates. Then, they can connect to a GitHub repository and then manage the configuration and deployment of their app before returning to the main home screen. Viewing a deployed app will allow a user to see documentation, run security scans, and view metrics like website traffic, CPU usage, logs, errors, etc.



UI Mockups

Deployed App Overview

 Vernal

App Name ☆

Owner Organization Lifecycle experimental

Search

OverviewMetricsNetwork GraphsVulnerabilitiesService Mesh

HomeCreateDocsSocial Hub

Settings

About

Source CodeView Code →

TechnologiesView Technologies →

Description

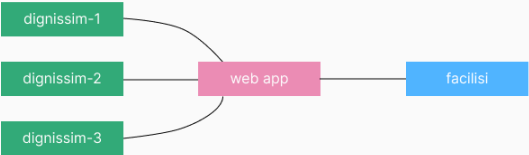
Owner Organization Lifecycle experimental

Type Website Tags sed,risus,ultrices,ultrices

App Alerts

No Alerts Right Now

Relations




View All Graphs →

Pods

Name	Node	Status	Restarts	Age
✓ vestibulum-aaccumsan	quam	Running	0	36 minutes
✓ amet-vulputate	quam	Running	0	36 minutes
✓ neque-vehicula	quam	Running	0	10 hours
✓ hendrerit-quis	quam	Running	0	10 hours
✓ risus-enim	quam	Running	0	10 hours
✓ commodo-tortor	quam	Running	0	15 hours

Deploying App View

 Vernal

Search

Home

Create

Docs

Social Hub

Settings

Deploying App Name

Fetch Repo

Publish

Register

CANCEL

START OVER

Logs

2024-02-08T04:52:10.000Z Beginning step Fetch Base

2024-02-08T04:52:10.000Z info: Fetching template content from remote URL

2024-02-08T04:52:10.000Z info: Listing files and directories in template

2024-02-08T04:52:10.000Z info: Processing 3 template files/directories with input values {"name":"sggsfd"}

2024-02-08T04:52:10.000Z info: Writing file catalog-info.yaml to template output path with mode 33188.

2024-02-08T04:52:10.000Z info: Writing file index.js to template output path with mode 33188.

2024-02-08T04:52:10.000Z info: Writing file package.json to template output path with mode 33188.


2024-02-08T04:52:10.000Z info: Template result written to /tmp/3f31d072-6fd7-49fc-9f14-29e57ff731ee

2024-02-08T04:52:10.000Z Finished step Fetch Base

2024-02-08T04:52:10.000Z Beginning step Publish

Search

Template Catalog Page

 Vernal

Create A New App

Search

Home

Create

Docs

Social Hub

Settings

Templates

Next.js App w/ FastAPI

A template for deploying Next.js

Choose

Next.js App w/ Express

A template for deploying Next.js

Choose

Next.js App w/ ApolloGraphQL

A template for deploying Next.js

Choose

Search and Filters

Search

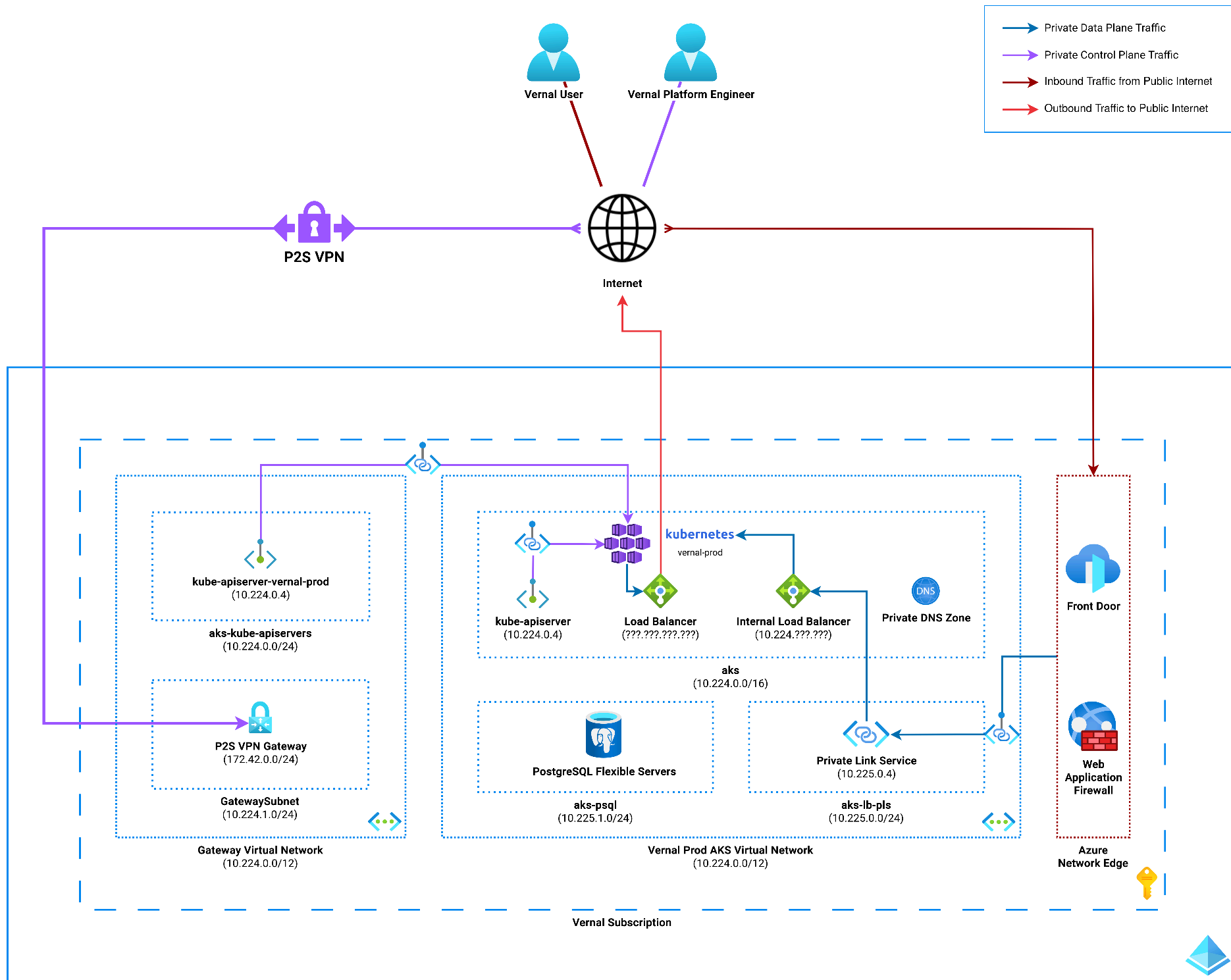
Categories

System Architecture Design Diagram

In production, Vernal will be deployed to Azure using their managed Kubernetes service, AKS. AKS will be run in private networking mode to ensure that the control plane is not exposed to the public internet. Inbound HTTP traffic for applications deployed in the Kubernetes cluster will go through Azure Front Door with Web Application Firewall enabled to provide intrusion detection/prevention and to protect against attacks. Front Door is privately networked through a private link to an internal load balancer pointing at an Istio Ingress Gateway deployed on Kubernetes. This is how end-users of deployed applications will be able to securely access them. Additionally, access to the Kubernetes API for platform engineers will be provided through a VPN connection to a virtual network that contains a private link to the AKS control plane.

We may also deploy our PostgreSQL databases to Azure, with them privately networked to a subnet through the Azure Backbone. However, we may instead decide to deploy PostgreSQL databases in-cluster for local development environments to avoid additional cloud costs.

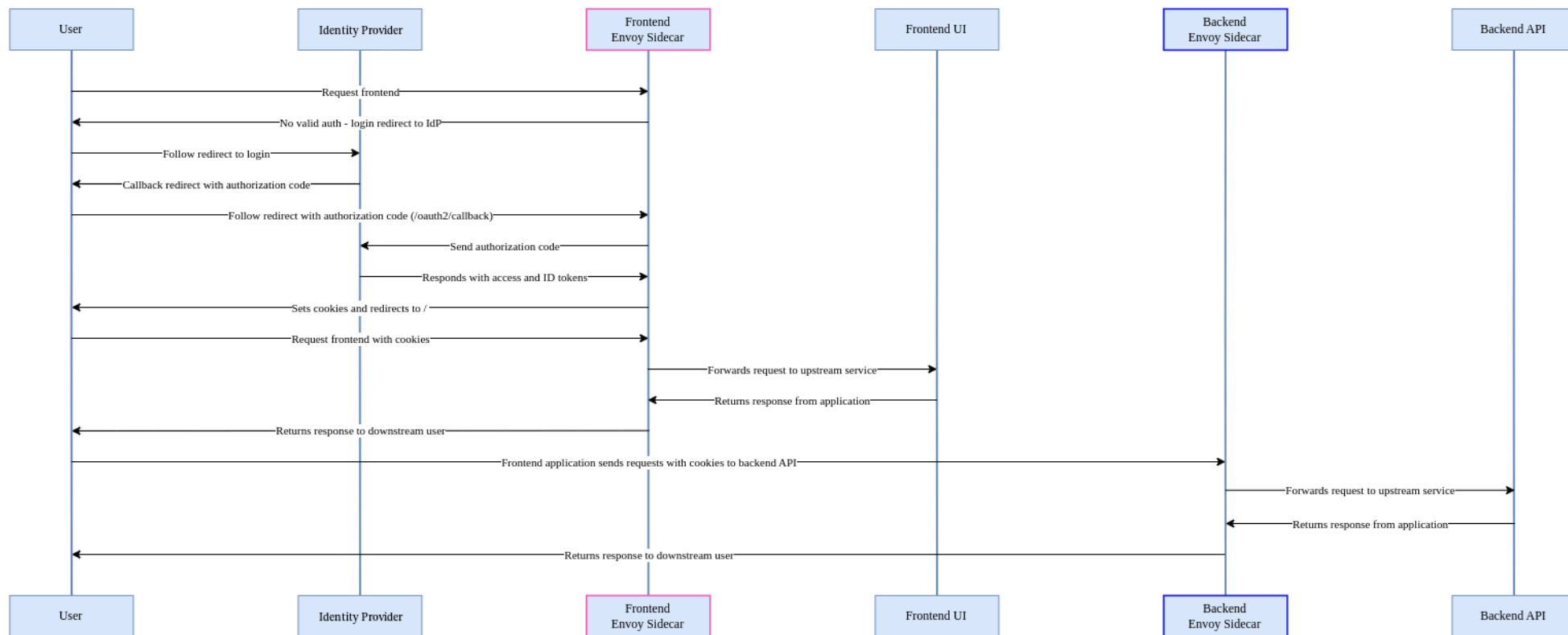
See the diagram below.



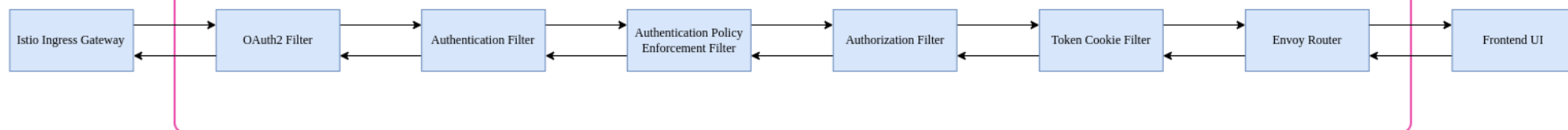
Sequence Diagram of Vernal Torii

Torii, our one-click authentication service, allows developers to easily enable OIDC authentication for their applications deployed on Vernal. Out of the box, Torii supports enabling authentication for full-stack applications, protecting the frontend and backend through the usage of Envoy filters. Once Torii is enabled for a particular application, several filters are injected into the Envoy proxies fronting its frontend and backend. The frontend and backend will require a token cookie with a particular role in order to access it, performing authorization checks. In addition to authorization, for the frontend, authentication will be implemented through OIDC with Keycloak. This is all possible because applications are deployed into a service mesh managed by Istio, meaning that each pod will have an Envoy proxy deployed in front of it automatically. For the developer, this all done with just a few clicks!

See the diagram below.



Frontend Envoy Sidecar Network Filter Chain



Backend Envoy Sidecar Network Filter Chain

