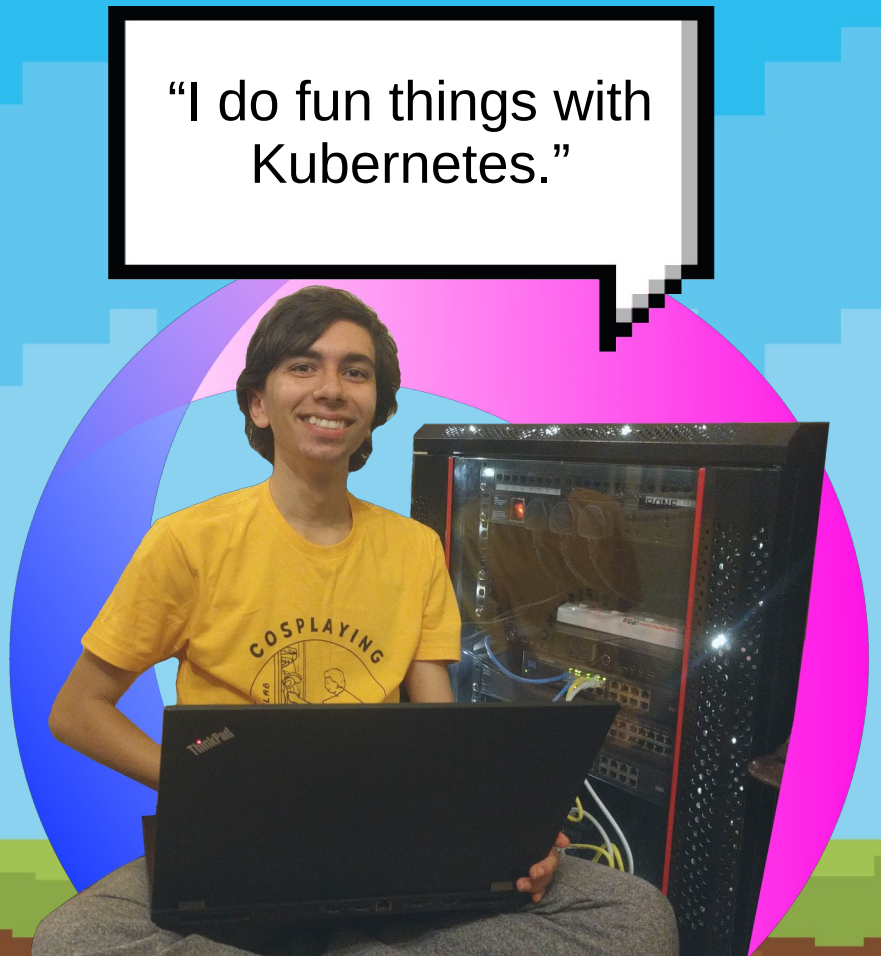
Krafting a FOSS Cloud Platform

By Alex "not a developer" Bissessur

alex.yaml

```yaml
---
apiVersion: v1
kind: Person
metadata:
 name: Alex Bissessur
spec:
  work:
    company: La Sentinelle
    role: Kubernetes Person
    location: Mauritius
  contact:
    website: alexbissessur.dev
    mastodon: moris.social/@AlexB
    github: github.com/xelab04
  interests:
    - Kubernetes
    - Linux
    - Free & Open Source Software
  hobbies:
    - Playing kubectl with Homelab
```



What is Kraft?

What is Kraft?

- A company which makes cheese.

DevCon 2024

- I wanted to host a workshop on Kubernetes
- Attendees bring laptops, tablets, phones.
- OS varies from Windows, MacOS, Linux etc…
- Workshops need a consistent environment
  (or you spend more time troubleshooting)
- The cloud is needed, but at what cost?

Rancher’s HobbyFarm

- I used HobbyFarm at SUSE Day at KC
- FOSS, and self-hostable
- Deploys VMs to a CSP
- Sets up lab environment
- Gives users a shell in the website
- Everything included, except CSP credits

Virtual Clusters

- KinD – run Kubernetes in Docker, perfect for testing
- Virtual clusters can be deployed to an existing K*s cluster
- Existing options:
  - VCluster, Loft Labs
  - K3k, Rancher/SUSE

K3k and VCluster

- Provision virtual clusters on a host
- Virtual cluster lives in pods, with the API server exposed
- Option to passthrough host resources, such as: storageclass, ingressclass, configmaps
- Virtual clusters are isolated across different namespaces

KRaft

- Kraft is built on K3k
  (also known as SUSE virtual clusters)
- Ingress is handled by Traefik*
- Storage by Longhorn*
- K3k offers two main features:
  - clusters
  - virtualclusterpolicies

How is KRaft Made?

- Kraft is split into ‘microservices’
- Auth service for login, registration, pwd reset
- Cluster service to provision clusters, expose the API, and retrieve kubeconfig file
- Resource service to retrieve cluster metrics
- Frontend of static website with Tailwind & Alpine
- MariaDB

Microservices

22 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

In software engineering, a **microservice** architecture is an architectural pattern that organizes an application into a collection of loosely coupled, fine-grained services that communicate through lightweight protocols. This pattern is characterized by the ability to develop and deploy services independently, improving modularity, scalability, and adaptability. However, it introduces additional complexity, particularly in managing distributed systems and inter-service communication, making the initial implementation more challenging compared to a monolithic architecture.[1]

Definition [edit]

There is no single, universally agreed-upon definition of microservices. However, they are generally characterized by a focus on modularity, with each service designed around a specific business capability. These services are loosely coupled, independently deployable, and often developed and scaled separately, enabling greater flexibility and agility in managing complex systems. Microservices architecture is closely associated with principles such as domain-driven design, decentralization of data and governance, and the flexibility to use different technologies for individual services to best meet their requirements. [2][3][4]

KRaft

```
NAME                                    CPU(cores)   MEMORY(bytes)
kraft-auth-7c4646b85c-m76jf             1m           5Mi
# written in Rust
kraft-cluster-manage-6cf9d74c9b-hnrmv   1m           9Mi
# also written in Rust
kraft-db-8566c5b599-4mtrb               1m           176Mi
# simple MariaDB pod
kraft-frontend-68d8c7698c-z5nck         0m           7Mi
# nginx with static files
kraft-resource-manage-7d7d979fc4-5qlwz  1m           290Mi
# python container, Flask
```

k3k-rs

- Kube-rs is the crate for interacting with Kubernetes resources from Rust
- Using K3k CRDs properly meant implementing them as Rust structs
- K3k-rs is an offshoot project
  - CRDs reimplemented
  - Convenience fns to manage K3k-related resources

List k3k clusters

```rust
let client = Client::try_default().await?;
let list: Vec<cluster::Cluster> = cluster::list::namespaced(&client, "k3k-namespace").await?;
```

```rust
let client = Client::try_default().await?;

let cluster_schema = k3k_rs::cluster::Cluster {
    metadata: kube::core::ObjectMeta {
        name: Some("test-cluster".to_string()),
        namespace: Some("k3k-namespace".to_string()),
        ..Default::default()
    },
    spec: ClusterSpec {
        // servers: 1, (default)
        // agents: 0, (default)
        // mode: "shared".to_string(), (default)
        // persistence: Some(PersistenceSpec {
        //     r#type: Some("dynamic".to_string()),
        //     storage_class_name: None,
        //     storage_request_size: Some("1G".to_string()),
        // }),
        expose: Some(ExposeSpec {
            LoadBalancer: Some(ExposeLoadBalancer {
                etcd_port: Some(2379),
                server_port: Some(443),
            }),
            NodePort: None,
            Ingress: None,
        }),
        ..Default::default()
    },
    status: None,
};

cluster::create(&client, namespace, &cluster_schema).await?;
```

```
197 + // Inside src/user.rs
198 +
199 + // ... (all other necessary imports)
200 +
201 + // The delete handler function MUST be public
202 + #[delete("/auth/user/account")]
203 + pub async fn delete_account( /* ... */ ) -> HttpResponse {
204 +     // ... implementation ...
205 + }
```

xelab04 on Oct 26 — Owner

Ah yes, fascinating. The parameters are comments, and the implementation is a comment saying "implementation". Not sure what I'm looking at here.

2

#hacktoberfest
#aislop

```
6     use actix_web::{middleware, web, App, HttpServer};
7   + use std::sync::Arc;
8   + // NEW: Import the KubeClient and anyhow::Result for error handling
9   + use k3k_rs::client::Client as KubeClient;
```

xelab04 1 hour ago — Owner

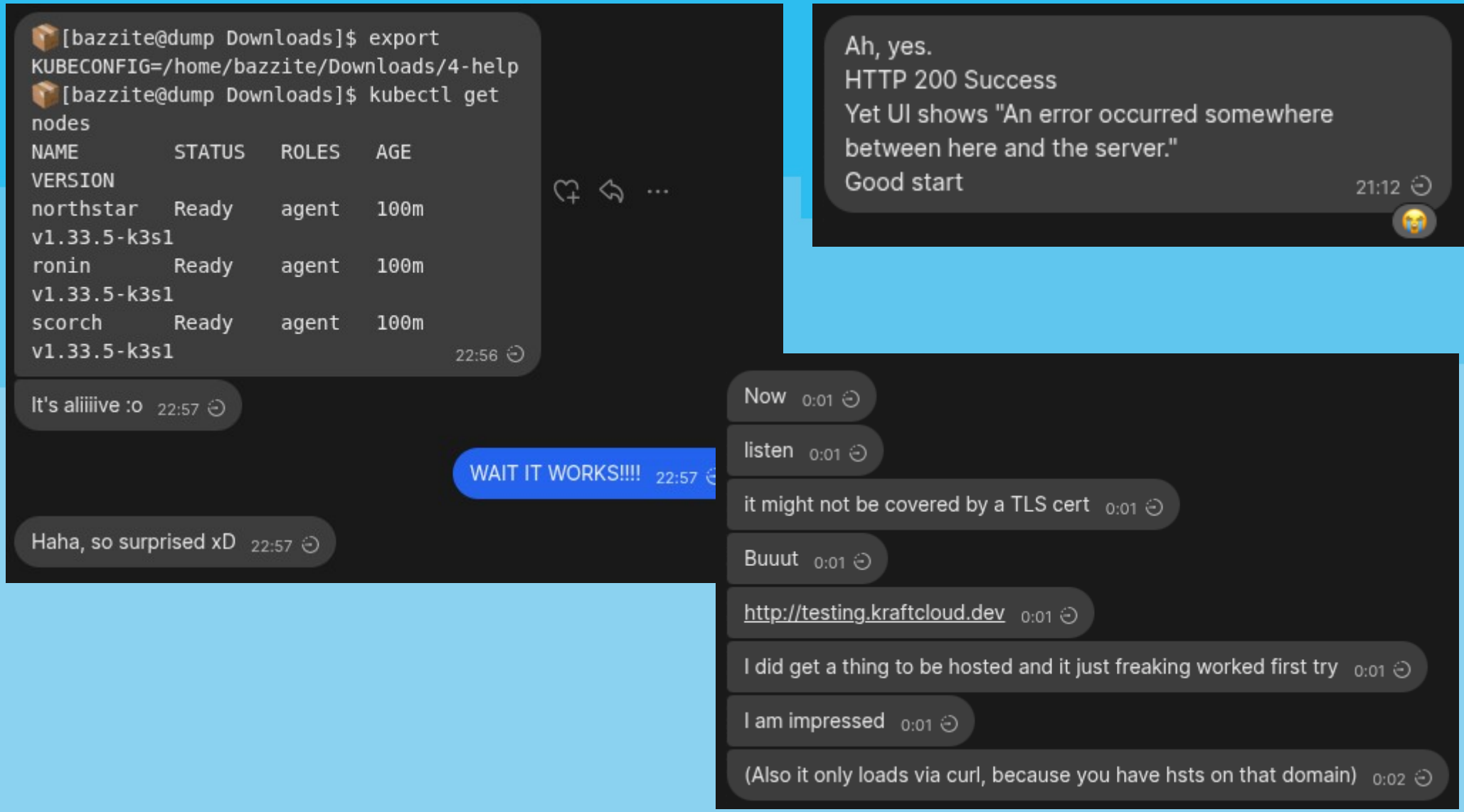
I guess AI does not, in fact, know about a niche Rust package I made myself.

Keeping Workloads Safe

- Users interact with only their virtual API
- Clusters are separated by namespace
- Each cluster has its own CoreDNS
- Clusters are isolated through network policies
- Security can be enhanced with VirtualClusterPolicies

VirtualClusterPolicies

- VCPs provide more control over K3k clusters
- Resource quotas
- Default resource limits/requests
- Allowed Mode
- What resources to sync
- PodSecurityAdmissionLevel
- PriorityClass
- Max resource limits/requests
- Disabling network policy


📦[bazzite@dump Downloads]$ export KUBECONFIG=/home/bazzite/Downloads/4-help
📦[bazzite@dump Downloads]$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
northstar Ready agent 100m v1.33.5-k3s1
ronin Ready agent 100m v1.33.5-k3s1
scorch Ready agent 100m v1.33.5-k3s1
22:56
It's aliiiive :o 22:57
WAIT IT WORKS!!!! 22:57
Haha, so surprised xD 22:57
Ah, yes.
HTTP 200 Success
Yet UI shows "An error occurred somewhere between here and the server."
Good start
21:12
Now 0:01
listen 0:01
it might not be covered by a TLS cert 0:01
Buuut 0:01
http://testing.kraftcloud.dev 0:01
I did get a thing to be hosted and it just freaking worked first try 0:01
I am impressed 0:01
(Also it only loads via curl, because you have hsts on that domain) 0:02

Work In Progress

- Ingresses are across the cluster – can cause conflicts
- Improvements needed on k3k-rs CRD structure
- K3k does not run in restricted mode
- Need to figure out backups
- Portability with Helm charts to install anywhere

Demo Time

Thank You!

alexbissessur.dev

t.me/alexbissessur

moris.social/@AlexB