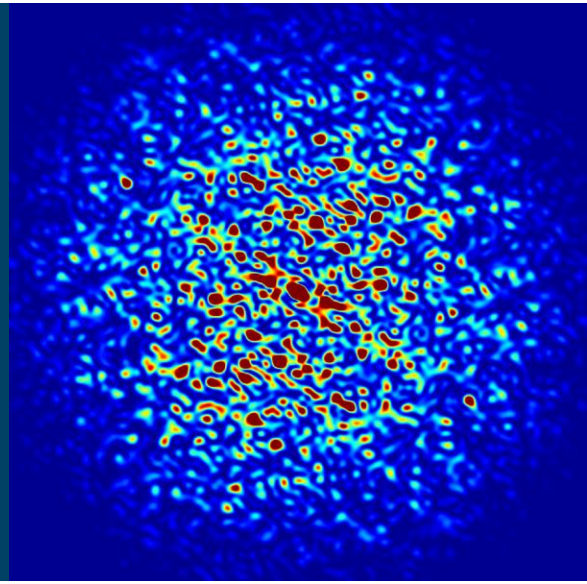
ACORN Nameserver Design Proposal



Elaine Chandler
Controls Group / Accelerator Systems Division
Advanced Photon Source
Argonne National Laboratory
May 19, 2025

Functional Requirements

- Central service for storing metadata describing information about the controls system, such as device definitions, properties or data channels, alarm configurations, node configuration, access control configuration, and property configurations
- Use secure gRPC as the primary protocol for service communication
- Uses PostgreSQL for storage
- Uses Keycloak integration for authentication and role-based access control (RBAC)

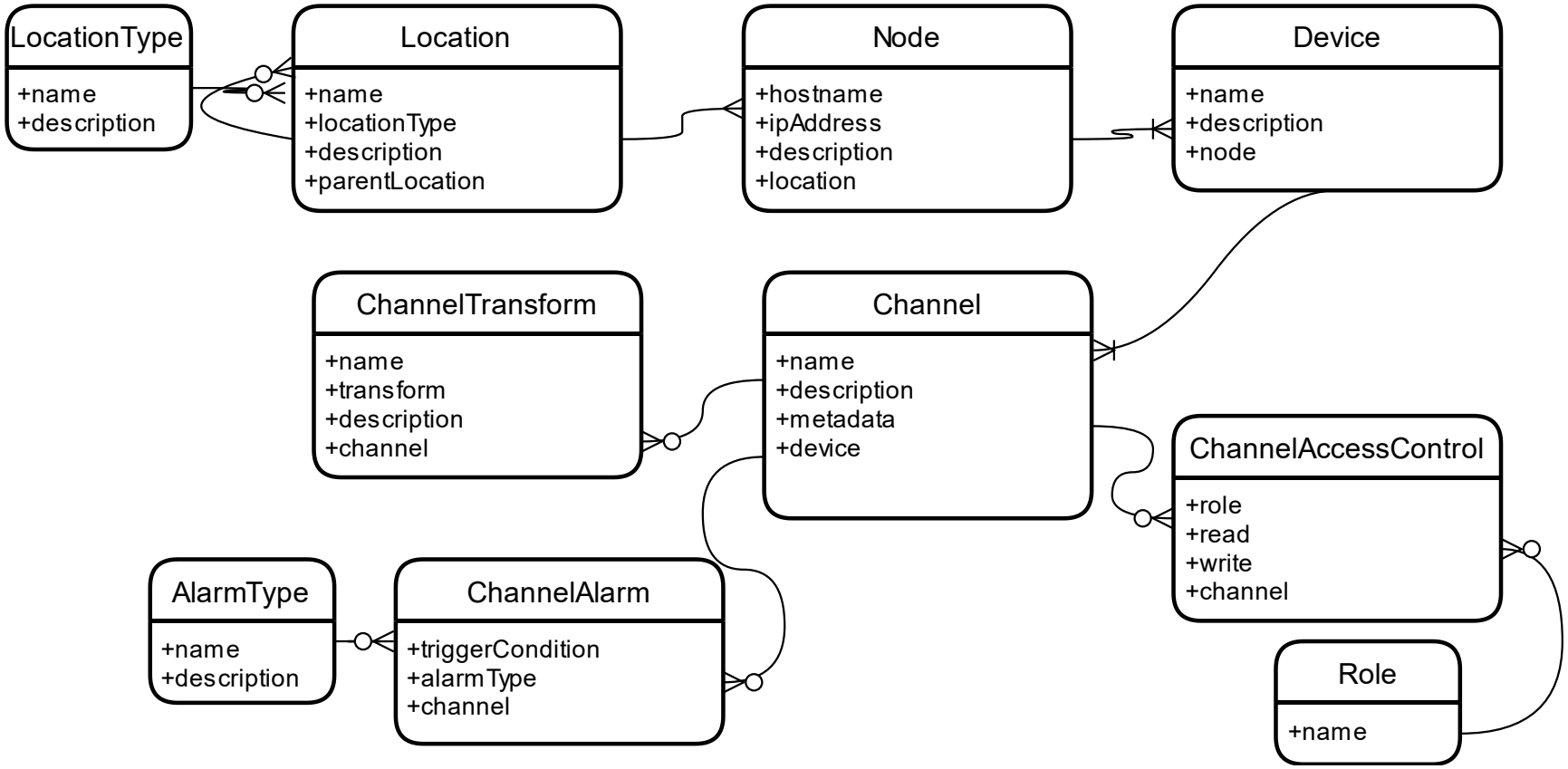
Overview

- Features summary
  - Secure gRPC API for managing and viewing control system information
  - PostgreSQL data storage
  - Integration with Keycloak for role-based access control
  - Client tool for interacting with service
- Outline
  - Review Nameserver functional API and system design
  - Demo Nameserver prototype

Main data components

- Location – physical location
- Node – VM or host running a “device”
- Device – entity that is a source/container of “channels”
- Channel – data globally accessible by an address/name. ex. properties, EPICS PVs
  - Includes transforms, alarm settings, access control configuration

Data model


LocationType
+name
+description
Location
+name
+locationType
+description
+parentLocation
Node
+hostname
+ipAddress
+description
+location
Device
+name
+description
+node
ChannelTransform
+name
+transform
+description
+channel
Channel
+name
+description
+metadata
+device
ChannelAccessControl
+role
+read
+write
+channel
AlarmType
+name
+description
ChannelAlarm
+triggerCondition
+alarmType
+channel
Role
+name

gRPC API

- Provide create, update, get, list, and delete methods for each unique data entity (Location, Node, Device, Channel, LocationType, AlarmType, and Role)
- Example:

```
message Device {
  string name = 1; //unique ID
  optional string description = 2;
  string node_hostname = 3;
}

message DeviceResponse {
  Device device = 1;
}

message CreateDeviceRequest {
  Device device = 1;
}

rpc CreateDevice(CreateDeviceRequest) returns
(DeviceResponse);

message GetDeviceRequest {
  string name = 1;
}
rpc GetDevice(GetDeviceRequest) returns (DeviceResponse);
```

```
message ListDevicesRequest {
  optional string name = 1; //pattern match
  optional string node_hostname = 2;
  optional PaginationRequest pagination = 3;
}
message ListDeviceResponse {
  repeated Device devices = 1;
  PaginationResponse pagination = 2;
}
rpc ListDevices(ListDevicesRequest) returns (ListDeviceResponse);

message UpdateDeviceRequest {
  Device device = 1;
}
rpc UpdateDevice(UpdateDeviceRequest) returns
(DeviceResponse);

message DeleteDeviceRequest {
  string name = 1;
}
rpc DeleteDevice(DeleteDeviceRequest) returns
(google.protobuf.Empty);
```

gRPC Channel API

Channel API includes extra API for managing channel transforms, alarm, and access control configurations

```
message Channel {
  string name = 1;
  optional string description = 2;
  string device_name = 3;
  optional string metadata = 4;
  repeated ChannelTransform transforms=5;
  repeated ChannelAlarm alarms = 6;
  repeated ChannelAccessControl accesscontrols = 7;
}

message ChannelTransform {
  string name = 1;
  string transform = 2;
  string description = 3;
}
message ChannelAlarm {
  string type = 1;
  string trigger_condition = 2;
}
message ChannelAccessControl {
  string role = 1;
  optional bool read = 2;
  optional bool write = 3;
}
```

```
rpc CreateChannel(CreateChannelRequest) returns (ChannelResponse);
rpc GetChannel(GetChannelRequest) returns (ChannelResponse);
rpc ListChannels(ListChannelsRequest) returns (ChannelListResponse);
rpc DeleteChannel(DeleteChannelRequest) returns (google.protobuf.Empty);
rpc UpdateChannel(UpdateChannelRequest) returns (ChannelResponse);


message AddChannelTransformRequest {
  string channel_name = 1;
  ChannelTransform transform = 2;
}
message ChannelTransformResponse {
  string channel_name = 1;
  ChannelTransform transform = 2;
}
rpc AddChannelTransform(AddChannelTransformRequest) returns
(ChannelTransformResponse);
rpc DeleteChannelTransform(DeleteChannelTransformRequest) returns
(google.protobuf.Empty);
rpc UpdateChannelTransform(UpdateChannelTransformRequest) returns
(ChannelTransformResponse);
```

Java Framework

- Uses **Quarkus**, Java Framework tailored for applications running in Kubernetes environment
- Created in 2019 by Redhat,
- Features
  - Full stack framework that makes it easy to use common technologies and libraries for cloud-native development. Uses standard interfaces to make it easy to swap out different technologies.
  - Tuned to run as native applications on containers. Optimized to have fast start time and low memory footprint.
  - Single step container deployments on Kubernetes. Automatically generates Kubernetes resources and deployment files
  - Developer friendly features such as live coding and automatic provisioning and configuration of supporting services, automatic data schema update

Implementation

- Integration with PostgreSQL, uses Hibernate ORM
- Secure communication with SSL
- Integration with Keycloak using OpenID Connect configuration
  - Configured ”user” to have read permissions, “admin” to have write permissions, uses Java annotations for setting
  - `@RolesAllowed("admin")`
- Data schema versioning and automatic migration using Flyway
- Build as container image. Used Podman for testing, has support for Docker
  - ./mvnw package -Dnative -Dquarkus.native.container-build=true -Dquarkus.container-image.build=true

```
#Database setup
prod.quarkus.datasource.db-kind = postgresql
%prod.quarkus.datasource.username = quarkus
%prod.quarkus.datasource.password = quarkus
%prod.quarkus.datasource.jdbc.url =
jdbc:postgresql://localhost:5432/nameserver

#TLS setup
quarkus.http.ssl.certificate.files=ssl/server.crt
quarkus.http.ssl.certificate.key-files=ssl/server.key
quarkus.http.insecure-requests=disabled

#Keycloak setup
%prod.quarkus.oidc.auth-server-
url=http://localhost:8180/realms/quarkus
quarkus.oidc.client-id=backend-service
quarkus.oidc.credentials.secret=secret
```