zetta scale

Zenoh

Getting started

The Zenoh team

Last update: 09/03/2022

Zenoh APIs

COTS and embedded devices

x86 and ARM targets

Native libraries and bindings

Rust

- Native library
- Async support

C

- Binding based on Rust library
- Native library with zenoh-pico

Python

- Binding based on Rust library

Coming soon...

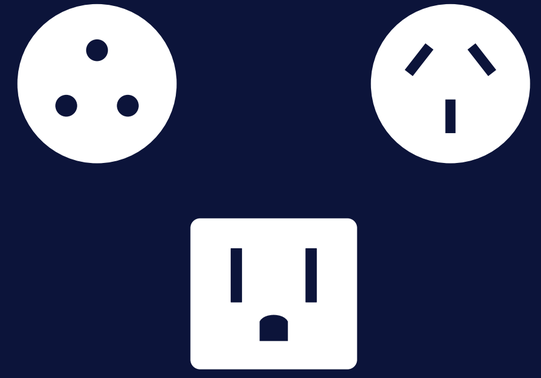
API bindings:
- C#
- Java
- Go

Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

Get your API

Use your preferred language



Python API

Python API

Easy install from PIP

Requires Python 3.6 minimum

Available on Pypi.org:

- Stable: https://pypi.org/project/eclipse-zenoh/
- Nightly: https://pypi.org/project/eclipse-zenoh-nightly/

Binary wheels for MacOS, Linux x86_64 and Linux i686

```
$ pip install eclipse-zenoh
```

```
$ pip install eclipse-zenoh-nightly
```

```
New API! -> Let's use it!
```

Note: Tutorial using Zenoh v0.5.0-beta9

Python API

Source distribution for any platform

Install the Rust toolchain: it is required to build the binding from the Rust version

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
$ source $HOME/.cargo/env
```

Compile and install the python API

```
$ apt install python3-pip python3-launchpadlib python3-testresources
$ git clone https://github.com/eclipse-zenoh/zenoh-python
$ cd zenoh-python
$ pip3 install -r requirements-dev.txt
$ python3 setup.py develop
```

Python API - Docs and examples

More information on python API is available at:

https://github.com/eclipse-zenoh/zenoh-python

Examples are available at:

https://github.com/eclipse-zenoh/zenoh-python/tree/master/examples

Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

C API

C API - zenoh-c

A binding on zenoh-rust

Requires C11 minimum

Stable available on Eclipse downloads:

- https://download.eclipse.org/zenoh/zenoh-c/latest/

Nightly to compile and install from source:

```
$ git clone https://github.com/eclipse-zenoh/zenoh-c.git
$ cd zenoh-c
$ mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release ..
$ cmake --build .
$ cmake --build . --target install
```

$ New API! -> Let's use it!

Note: Tutorial using Zenoh v0.5.0-beta9

C API - zenoh-pico

A native C library for embedded systems

Requires C99 minimum

Stable available on Eclipse downloads:

- https://download.eclipse.org/zenoh/zenoh-pico/latest/

Nightly to compile and install from source:

```
$ git clone https://github.com/eclipse-zenoh/zenoh-pico
$ cd zenoh-pico
$ make
$ make install
```

Note: Tutorial using Zenoh v0.5.0-beta9

C API - Docs and examples

More information on C API is available at:

https://github.com/eclipse-zenoh/zenoh-c

https://github.com/eclipse-zenoh/zenoh-pico

Examples are available at:

https://github.com/eclipse-zenoh/zenoh-c/tree/master/examples

https://github.com/eclipse-zenoh/zenoh-pico/tree/master/examples/net

Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

Rust API

Rust API

Requires Rust 1.51 minimum

Include it in your Cargo.toml

Stable available on crates.io:

- https://crates.io/crates/zenoh

Nightly available from source:

```
# Stable
[dependencies]
async-std = "1.10.0"
env_logger = "0.9.0"
zenoh = "0.5.0-beta.9"
```

```
# Nightly
[dependencies]
async-std = "1.10.0"
env_logger = "0.9.0"
zenoh = { git = "https://github.com/
eclipse-zenoh/zenoh.git" }
```

```
$ New API! -> Let’s use it!
```

Note: Tutorial using Zenoh v0.5.0-beta9

Rust API - Docs and examples

More information on Rust API is available at:

https://github.com/eclipse-zenoh/zenoh

Examples are available at:

https://github.com/eclipse-zenoh/zenoh/tree/master/examples

Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

Main concepts

Abstractions, primitives, communication models

Abstractions

Resource. A **named data**, in other terms a (key, value)

(e.g. (/home/kitchen/sensor/temp, 21.5),
(/home/kitchen/sensor/hum, 0.67))

Key expression. An **expression** identifying a set of keys

(e.g. /home/kitchen/sensor/*,
/home/**/temp)

Selector. An **expression** identifying a set of resources

(e.g. /home/*/sensor/air?co2>12[humidity])

Note: Tutorial using Zenoh v0.5.0-beta9

Abstractions

Publisher. A **spring** of values for a key expression

(e.g. /home/kitchen/sensor/temp, /home/kitchen/sensor/*)

Subscriber. A **sink** of values for a key expression

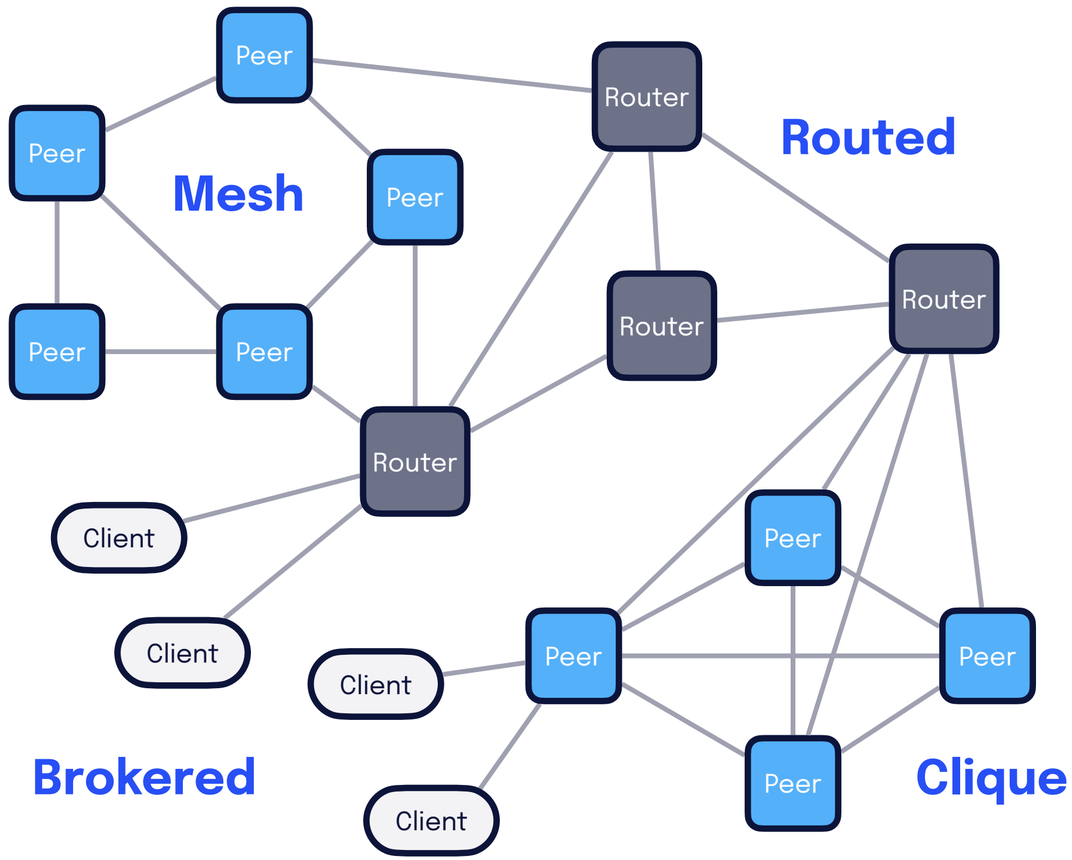
(e.g. /home/kitchen/sensor/temp, /home/kitchen/sensor/*)

Queryable. A **well** of values for a key expression

(e.g. /home/**)

Note: Tutorial using Zenoh v0.5.0-beta9

Communication models



Peer-to-peer

Clique and mesh topologies

Brokered

Clients communicate through a router or a peer

Routed

Routers forward data to and from peers and clients

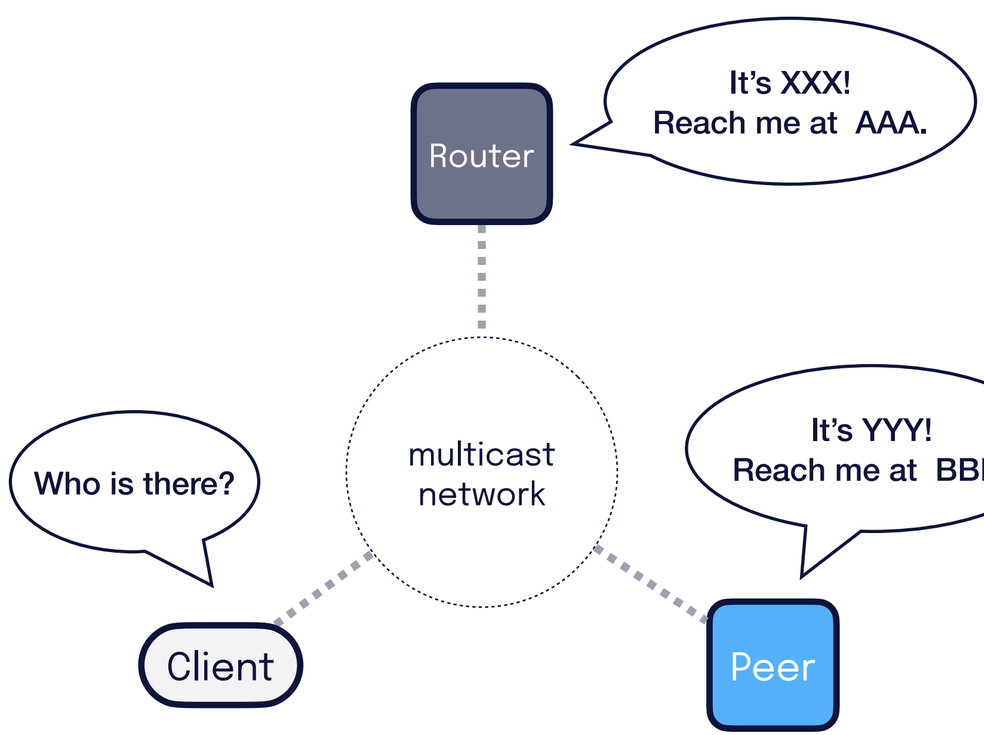
Note: Tutorial using Zenoh v0.5.0-beta9

Multicast scouting

Zenoh performs **scouting** to discover who is around, e.g. routers or peers

Scouting makes use of **multicast** capabilities of the underlying network

When a new zenoh node is discovered, then a **zenoh session** can be established



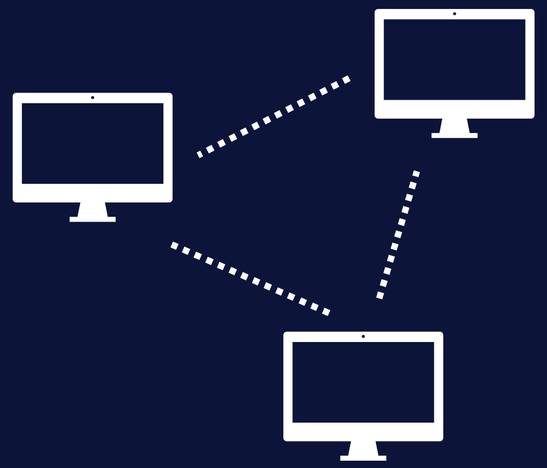

```
Default IP multicast group: 224.0.0.224
Default UDP scouting port:  7447
```

Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

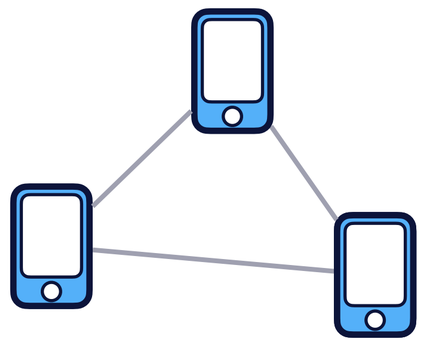
Peer-to-peer communication

Directly communicate



What is *peer-to-peer*?

Peer-to-peer is a **communication model** where zenoh peers **talk directly with each other** without the intervention of any zenoh infrastructure



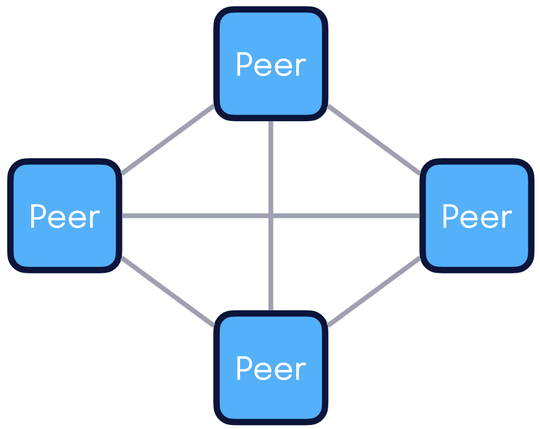
Note: Tutorial using Zenoh v0.5.0-beta9

Clique

All peers are **connected** to **each other**

Direct communication between peers

Clique is the **common case** for peer-to-peer

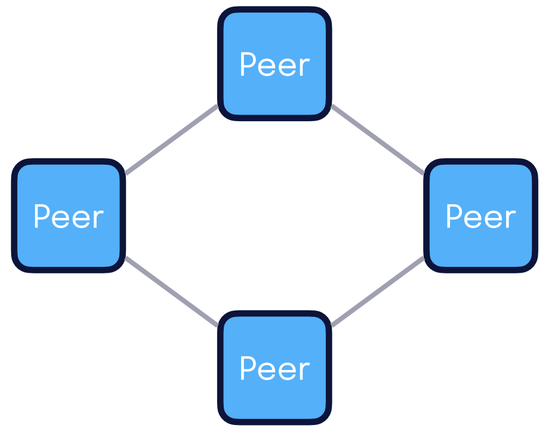


Mesh

Peers are **partially connected** to **each other**

Peers collaborate in the communication

Mesh supports **advanced** peer-to-peer **cases**

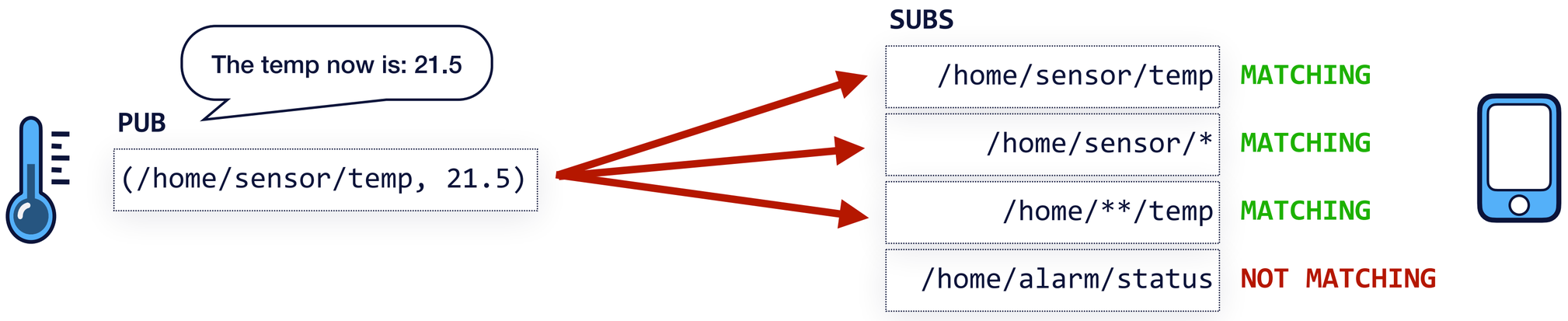


Note: Tutorial using Zenoh v0.5.0-beta9

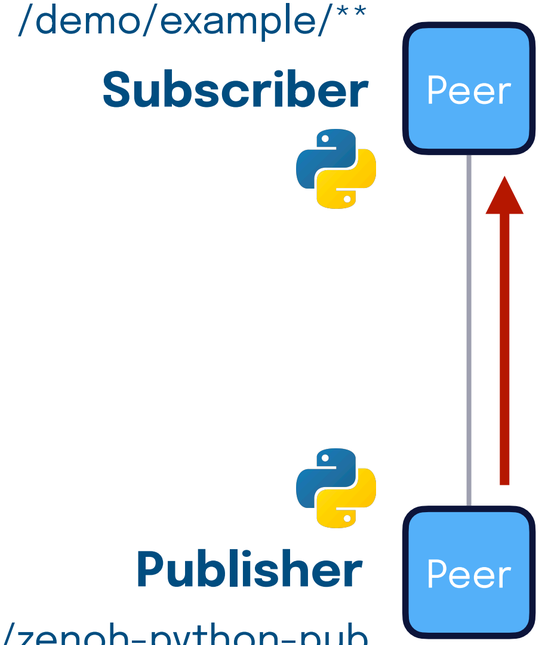
Pub/sub

Pub/sub, what for?

Distribute **values published** on a given **key expression** to all the **subscribers matching** that specific key expression



Note: Tutorial using Zenoh v0.5.0-beta9



Note: Tutorial using Zenoh v0.5.0-beta9

Publisher

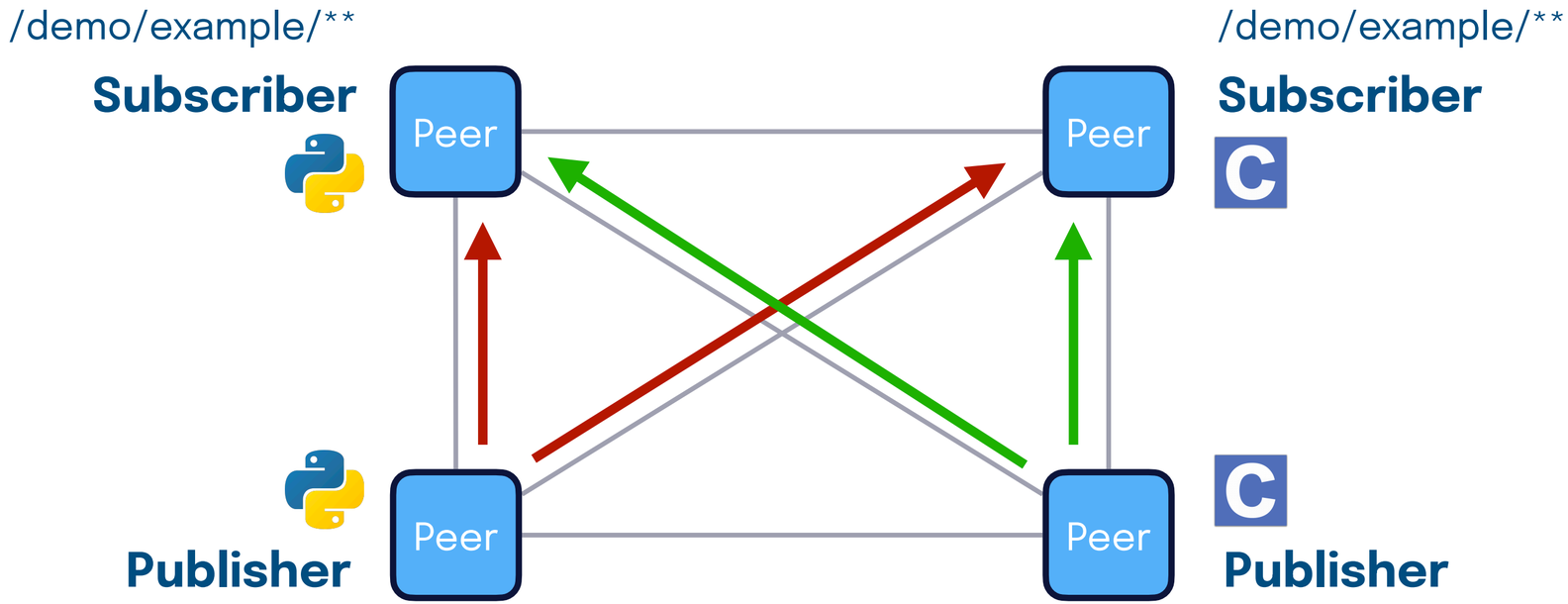
```python
import time
import zenoh

# initiate logging
zenoh.init_logger()

# configure zenoh in peer mode
conf = zenoh.Config.from_json5('{"mode": "peer"}')

# open a zenoh session
session = zenoh.open(conf)

# publish
key_expr = "/demo/example/zenoh-python-pub"
val = "Hello!"
while True:
    time.sleep(1.0)
    session.put(key_expr, val)
```

Subscriber

```python
import time
import zenoh

# initiate logging
zenoh.init_logger()

# configure zenoh in peer mode
conf = zenoh.Config.from_json5('{"mode": "peer"}')

# open a zenoh session
session = zenoh.open(conf)

# subscribe
def callback(sample):
    print("('{}': '{}')"
    .format(sample.key_expr, sample.payload))

key_expr = "/demo/example/**"
sub = session.subscribe(key_expr, callback)

while True:
    time.sleep(1.0)
```

Note: Tutorial using Zenoh v0.5.0-beta9

Pub example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_pub.py
Sub example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_sub.py



Note: Tutorial using Zenoh v0.5.0-beta9

Publisher

```c
#include <string.h>
#include <unistd.h>
#include <zenoh.h>

int main(int argc, char **argv) {
    // initiate logging
    z_init_logger();

    // open a zenoh session in peer mode
    z_owned_config_t config = z_config_default();
    z_config_set(z_loan(config), ZN_CONFIG_MODE_KEY, "peer");
    z_owned_session_t s = z_open(z_move(config));

    char buf[256];
    sprintf(buf, "%s", "Hello!");
    while (1) {
        // publish
        z_put(z_loan(s), z_expr("/demo/example/zenoh-c-pub")),
            (const uint8_t *)buf, strlen(buf));
        sleep(1);
    }

    return 0;
}
```

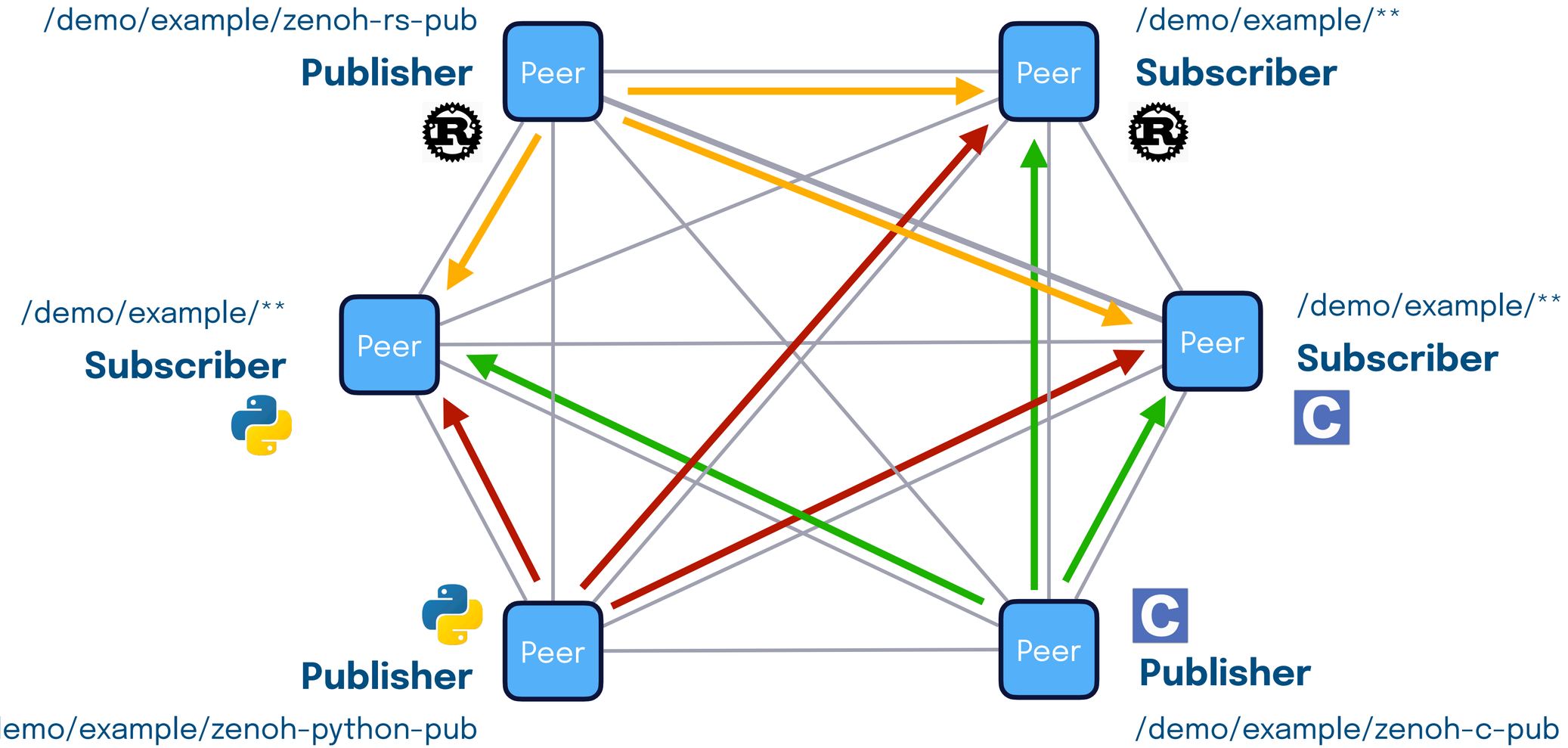
Subscriber

```c
#include <string.h>
#include <unistd.h>
#include <zenoh.h>

void callback(const z_sample_t *s, const void *arg) {
    printf("('%.*s': '%.*s')\n",
      (int)s->key.suffix.len, s->key.suffix.start,
      (int)s->value.len, s->value.start);
}

int main(int argc, char **argv) {
    // initiate logging
    z_init_logger();

    // open a zenoh session in peer mode
    z_owned_config_t config = z_config_default();
    z_config_set(z_loan(config), ZN_CONFIG_MODE_KEY, "peer");
    z_owned_session_t s = z_open(z_move(config));

    // subscribe
    z_owned_subscriber_t sub = z_subscribe(
         z_loan(s), z_expr("/demo/example/**"),
         z_subinfo_default(), callback, NULL);

    while (1) { sleep(1); }
    return 0;
}
```

C

Note: Tutorial using Zenoh v0.5.0-beta9

Pub example: https://github.com/eclipse-zenoh/zenoh-c/blob/master/examples/z_pub.c
Sub example: https://github.com/eclipse-zenoh/zenoh-c/blob/master/examples/z_sub.c



Note: Tutorial using Zenoh v0.5.0-beta9

Publisher

```
use async_std::task::sleep;
use std::time::Duration;
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;

#[async_std::main]
async fn main() {
    // initiate logging
    env_logger::init();

    // open a zenoh session in peer mode
    let mut config = Config::default();
    config.set_mode(Some(WhatAmI::Peer));
    let session = zenoh::open(config).await.unwrap();

    loop {
        // publish
        session.put("/demo/example/zenoh-rs-pub",
                    "Hello!").await.unwrap();
        sleep(Duration::from_secs(1)).await;
    }
}
```

Subscriber

```
use async_std::stream::StreamExt;
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;

#[async_std::main]
async fn main() {
    // initiate logging
    env_logger::init();

    // open a zenoh session in peer mode
    let mut config = Config::default();
    config.set_mode(Some(WhatAmI::Peer));
    let session = zenoh::open(config).await.unwrap();

    // subscribe
    let mut sub = session.subscribe("/demo/example/**")
                         .await.unwrap();

    while let Some(sample) = sub.receiver().next().await {
        println!("('{}': '{}')", sample.key_expr.as_str(),
            String::from_utf8_lossy(
                &sample.value.payload.contiguous()));
    }
}
```

Note: Tutorial using Zenoh v0.5.0-beta9

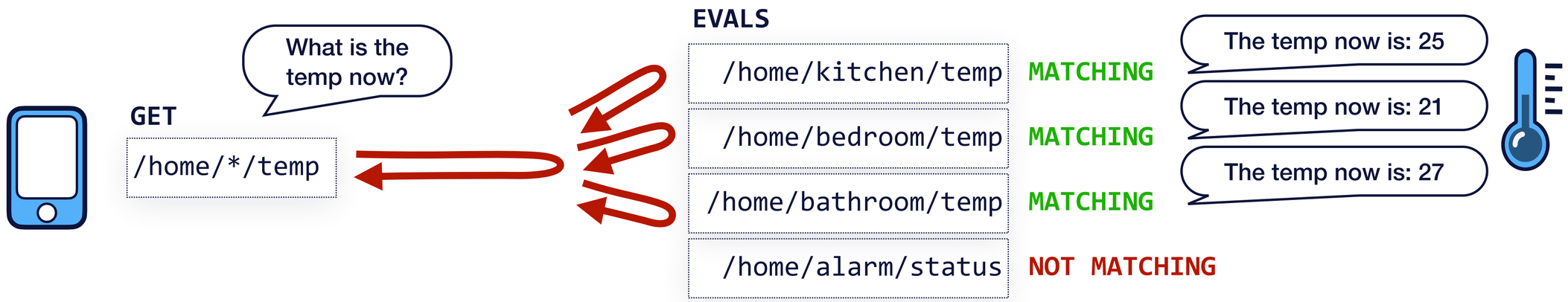
Pub example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_pub.rs
Sub example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_sub.rs

zetta scale

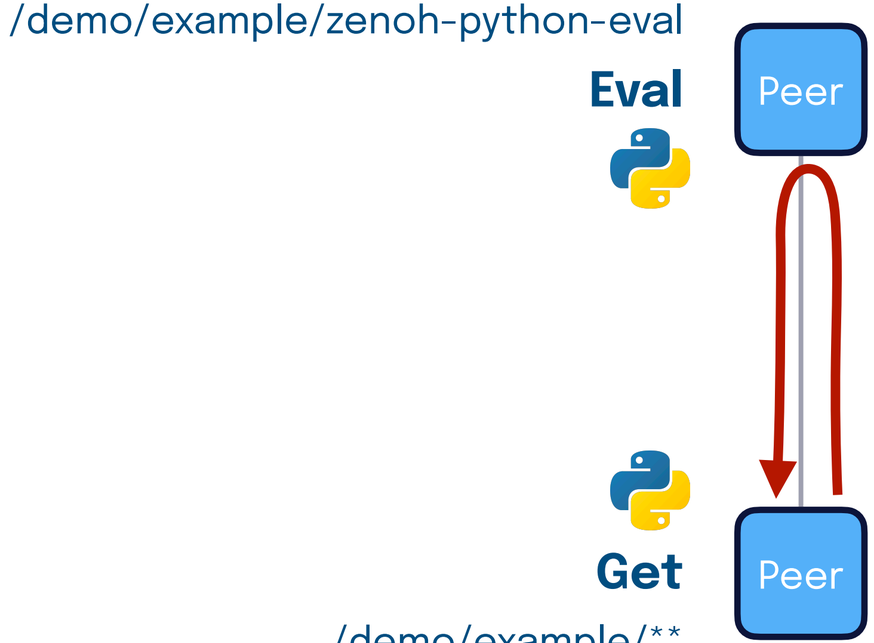
Distributed computed values

Distributed computed values, what for?

Request the **on-demand computation** of fresh **values** via a **distributed query** matching a given **key expression.** It is a **queryable**.



Note: Tutorial using Zenoh v0.5.0-beta9



Note: Tutorial using Zenoh v0.5.0-beta9

Get

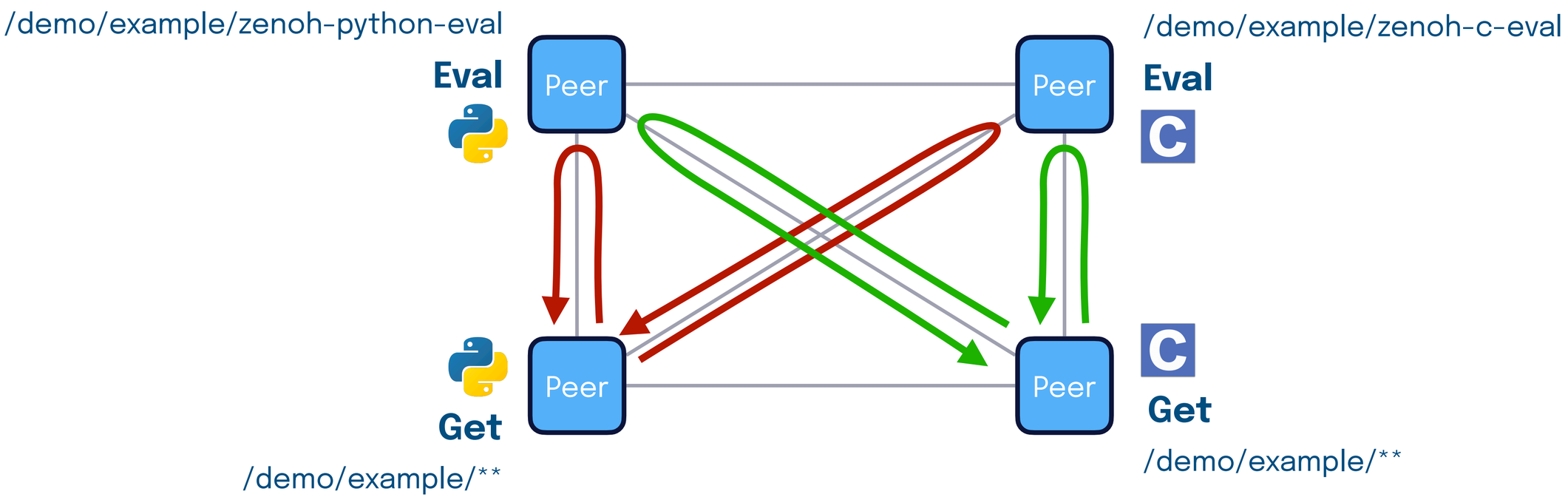
```
import zenoh

# initiate logging
zenoh.init_logger()

# configure zenoh in peer mode
conf = zenoh.Config.from_json5('{"mode": "peer"}')

# open a zenoh session
session = zenoh.open(conf)

# query
replies = session.get_collect("/demo/example/**")
for r in replies:
    print("('{}': '{}')".format(
        r.data.key_expr, r.data.payload))
```

Eval

```
import time
import zenoh
from zenoh import config, Sample
from zenoh.queryable import EVAL

# initiate logging
zenoh.init_logger()

# configure zenoh in peer mode
conf = zenoh.Config.from_json5('{"mode": "peer"}')

# open a zenoh session
session = zenoh.open(conf)

# eval
key_expr = "/demo/example/zenoh-python-eval"

def callback(query):
    query.reply(Sample(key_expr,
        payload="Eval from Python!".encode()))

queryable = session.queryable(key_expr, EVAL, callback)

while True:
    time.sleep(1.0)
```

Note: Tutorial using Zenoh v0.5.0-beta9

Get example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_get.py
Eval example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_eval.py



Note: Tutorial using Zenoh v0.5.0-beta9

Get

```c
#include <zenoh.h>

int main(int argc, char **argv) {
    // initiate logging
    z_init_logger();
    // open a zenoh session in peer mode
    z_owned_config_t config = z_config_default();
    z_config_set(z_loan(config), ZN_CONFIG_MODE_KEY, "peer ");
    z_owned_session_t s = z_open(z_move(config));
    // query
    z_query_target_t target = z_query_target_default();
    target.target.tag = z_target_t_ALL;

    z_owned_reply_data_array_t r = z_get_collect(z_loan(s),
           z_expr("/demo/example/**"), "", target,
           z_query_consolidation_default());

    for (unsigned int i = 0; i < r.len; ++i) {
        printf("('%.*s': '%.*s')\n",
            (int)r.val[i].data.key.suffix.len,
            r.val[i].data.key.suffix.start,
            (int)r.val[i].data.value.len,
            r.val[i].data.value.start);
    }

    return 0;
}
```

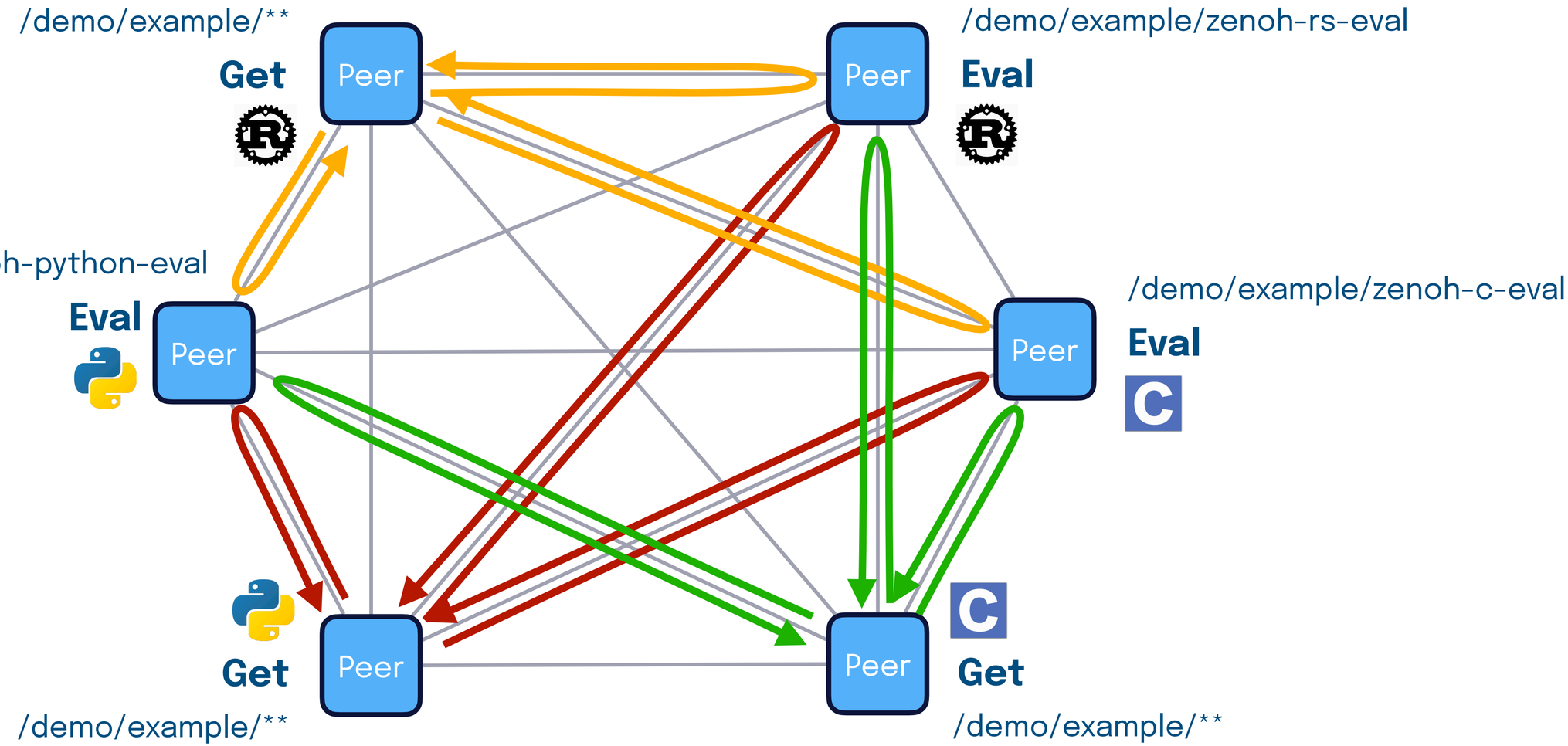
Eval

C

```c
#include <zenoh.h>

char *expr = "/demo/example/zenoh-c-eval";
char *value = "Eval from C!";

void callback(const z_query_t *query, const void *arg) {
    z_bytes_t res = z_query_key_expr(query).suffix;
    z_bytes_t pred = z_query_predicate(query);
    z_send_reply(query, expr,
            (const unsigned char *)value, strlen(value));
}

int main(int argc, char **argv) {
    // initiate logging
    z_init_logger();
    // open a zenoh session in peer mode
    z_owned_config_t config = z_config_default();
    z_config_set(z_loan(config), ZN_CONFIG_MODE_KEY, "peer");
    z_owned_session_t s = z_open(z_move(config));
    // eval
    z_owned_queryable_t qable = z_queryable_new(z_loan(s),
                z_expr(expr), ZN_QUERYABLE_EVAL, callback, NULL);

    while (1) { sleep(1); } return 0;
}
```

Note: Tutorial using Zenoh v0.5.0-beta9

Get example: https://github.com/eclipse-zenoh/zenoh-c/blob/master/examples/z_get.c
Eval example: https://github.com/eclipse-zenoh/zenoh-c/blob/master/examples/z_eval.c



Note: Tutorial using Zenoh v0.5.0-beta9

Get

```
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;

#[async_std::main]
async fn main() {
    // initiate logging
    env_logger::init();

    // open a zenoh session in peer mode
    let mut config = Config::default();
    config.set_mode(Some(WhatAmI::Peer));
    let s = zenoh::open(config).await.unwrap();

    // query
    let mut replies = s.get("/demo/example/**").await.unwrap();

    while let Some(reply) = replies.next().await {
        println!("('{}': '{}')",
           reply.data.key_expr.as_str(),
           String::from_utf8_lossy(&reply.data.value
               .payload.contiguous())
        );
    }
}
```

Eval

```
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;
use zenoh::prelude::Sample;
use zenoh::queryable::EVAL;

#[async_std::main]
async fn main() {
    // initiate logging
    env_logger::init();

    // open a zenoh session in peer mode
    let mut config = Config::default();
    config.set_mode(Some(WhatAmI::Peer));
    let session = zenoh::open(config).await.unwrap();

    // eval
    let key_expr: &str = “/demo/example/zenoh-rs-eval";
    Let value: &str = "Eval from Rust!";
    let mut qbl = session.queryable(key_expr)
                         .kind(EVAL).await.unwrap();

    while let Some(query) = qbl.receiver().next().await {
        query.reply(Sample::new(
            key_expr.clone(), value.clone()));
    }
}
```

Note: Tutorial using Zenoh v0.5.0-beta9

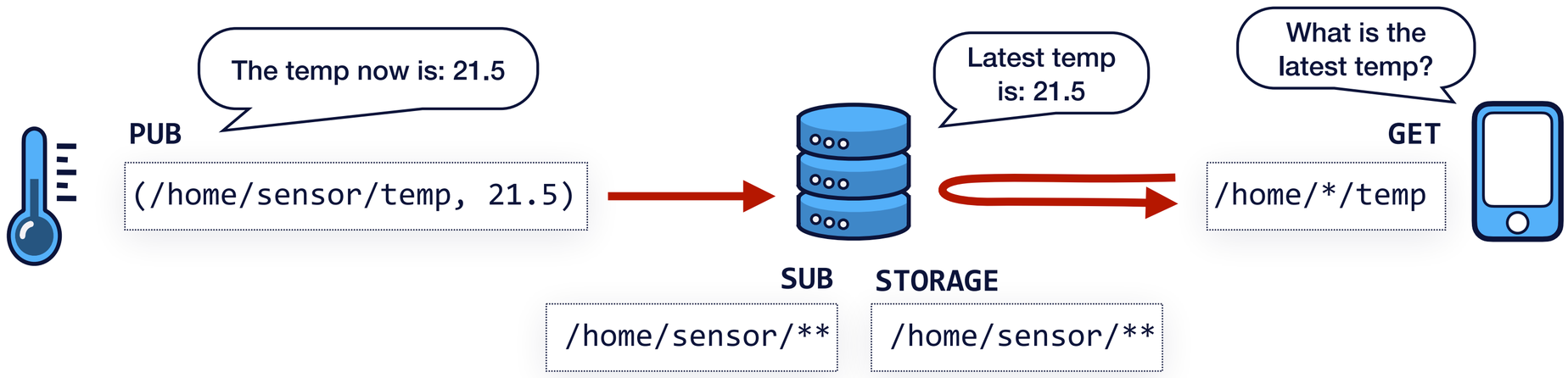
Get example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_get.rs
Eval example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_eval.rs

zetta scale

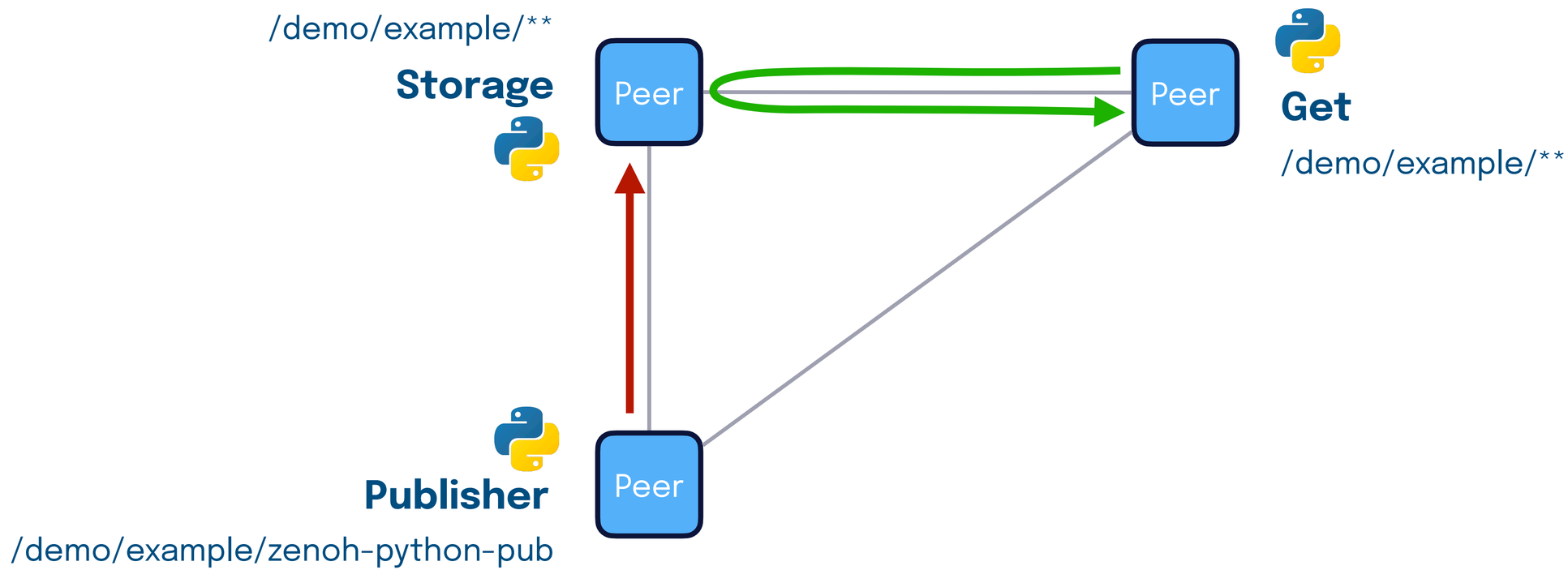
Distributed storages

Distributed storages, what for?

Store data as soon as it is published and allow to **retrieve** it **on-demand.** It is a **combination** of a **subscriber** and a **queryable**.



Note: Tutorial using Zenoh v0.5.0-beta9



Note: Tutorial using Zenoh v0.5.0-beta9

Storage

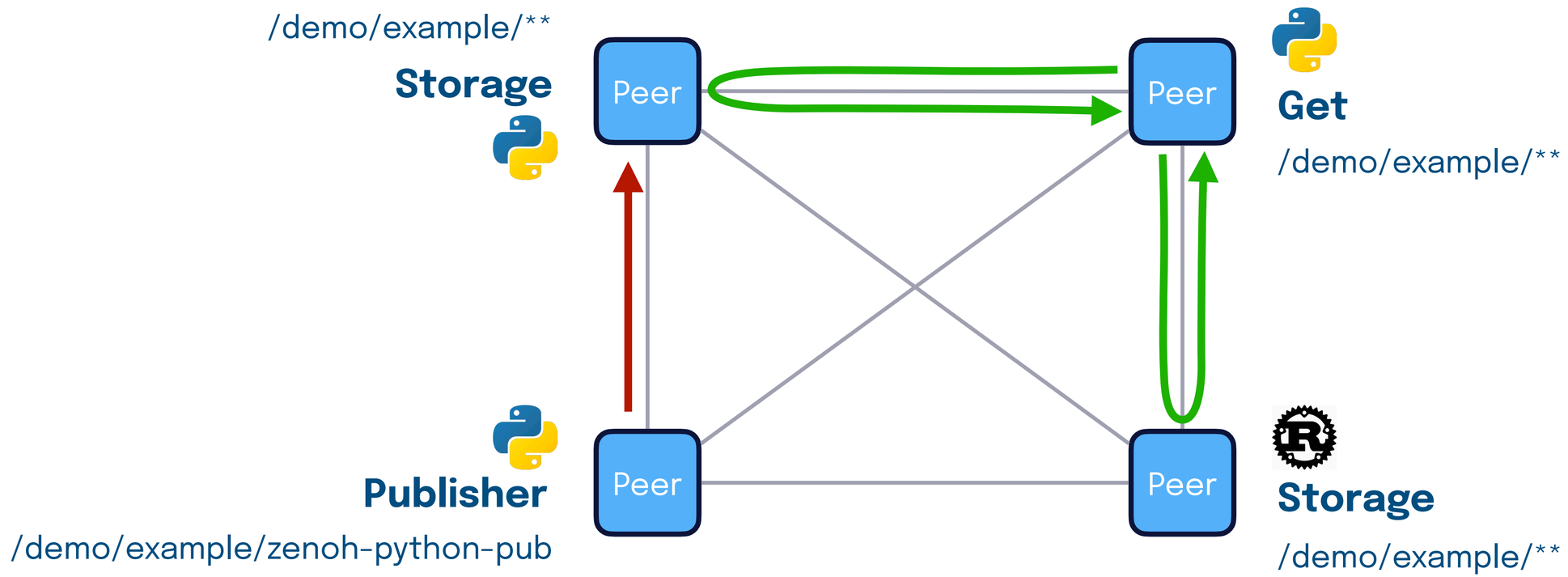
```
import time
import zenoh
from zenoh import config, Sample
from zenoh.queryable import STORAGE

# initiate logging
zenoh.init_logger()

# configure zenoh in peer mode
conf = zenoh.Config.from_json5('{"mode": "peer"}')
# open a zenoh session

session = zenoh.open(conf)
```

Continue… =>

```
store = {}

key_expr = “/demo/example/zenoh-python-eval"

# sub => insert/delete data into/from the store
def data_handler(sample):
    if sample.kind == SampleKind.DELETE:
        store.pop(str(sample.key_expr), None)
    else:
        store[str(sample.key_expr)] = (sample.value,
                                       sample.source_info)

subscriber = session.subscribe(key_expr, data_handler)

# storage => fetch data from the store and reply back
def query_handler(query):
    replies = []
    for stored_name, (data, source_info) in store.items():
        if KeyExpr.intersect(query.key_selector, stored_name):
            sample = Sample(stored_name, data)
            sample.with_source_info(source_info)
            query.reply(sample)

queryable = session.queryable(key_expr, STORAGE, query_handler)

while True:
    time.sleep(1.0)
```

Note: Tutorial using Zenoh v0.5.0-beta9

Storage example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_storage.py



Note: Tutorial using Zenoh v0.5.0-beta9

Storage

```rust
use futures::{prelude::*, select};
use std::collections::HashMap;
use std::time::Duration;
use zenoh::config::Config;
use zenoh::prelude::{config::WhatAmI, Sample, SampleKind};
use zenoh::queryable::STORAGE;
use zenoh::utils::key_expr;

#[async_std::main]
async fn main() {
  // initiate logging
  env_logger::init();

  // open a zenoh session in peer mode
  let mut config = Config::default();
  config.set_mode(Some(WhatAmI::Peer));
  let session = zenoh::open(config).await.unwrap();

  let key_expr: &str = "/demo/example/**";

  // subscribe
  let mut subscriber = session.subscribe(key_expr)
                        .await.unwrap();
  // declare the queryable for storage
  let mut queryable = session.queryable(key_expr).kind(STORAGE)
                      .await.unwrap();
```

Continue… =>

```rust
  let mut stored: HashMap<String, Sample> = HashMap::new();

  loop {
    select!(
      // sub => insert/delete data into/from the store
      sample = subscriber.next() => {
        let sample = sample.unwrap();
        if sample.kind == SampleKind::Delete {
          stored.remove(&sample.key_expr.to_string());
        } else {
          stored.insert(sample.key_expr.to_string(), sample);
        }
      },
      // storage => fetch data from the store and reply back
      query = queryable.next() => {
        let query = query.unwrap();
        for (stored_name, sample) in stored.iter() {
          if key_expr::intersect(query.selector()
                        .key_selector.as_str(), stored_name) {
            query.reply(sample.clone());
          }
        }
      },
    );
  }
}
```

```
# Update Cargo.toml
[dependencies]
futures = "0.3.21"
```

Note: Tutorial using Zenoh v0.5.0-beta9

Storage example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_storage.rs

zetta scale

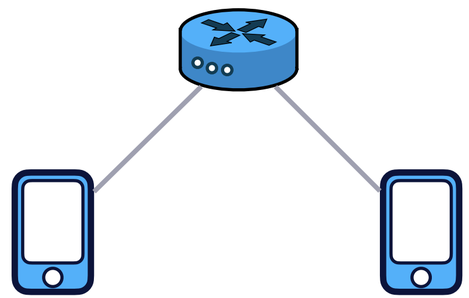
Brokered communication

Let’s use the zenoh router

What is *brokered*?

Brokered is a **communication model** where communication between zenoh peers and clients is **supported by** a **zenoh router**

The zenoh router comprises the **zenoh infrastructure**

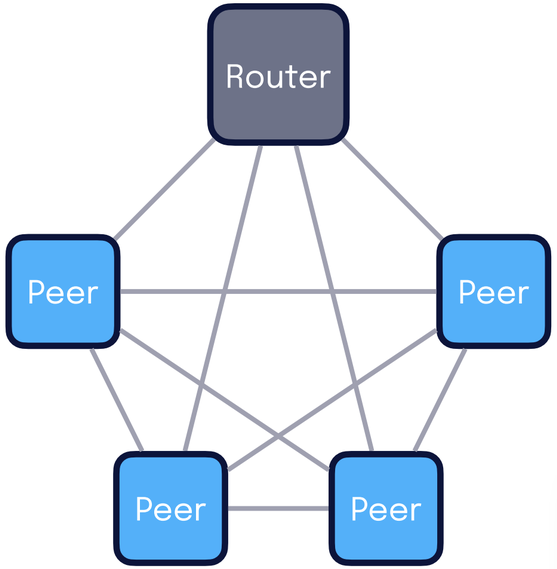


Note: Tutorial using Zenoh v0.5.0-beta9

Peer mode

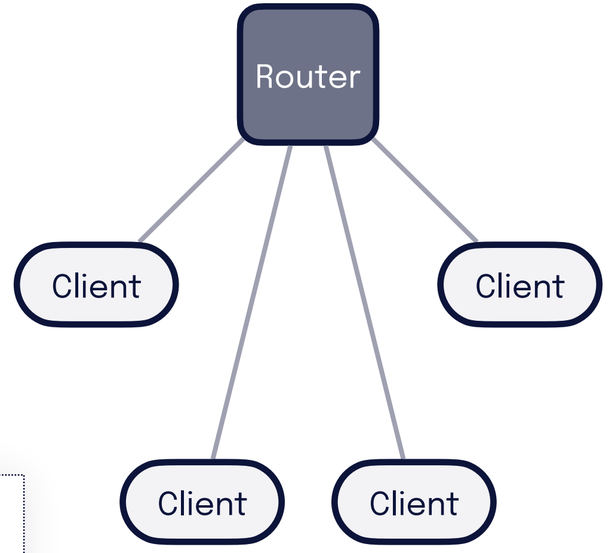
Local communication stays **peer-to-peer**

Remote communication via the **zenoh router**



Client mode

All communication via the **zenoh router**



Applications can operate in "client" or "peer" mode when connecting to a zenoh router

Note: Tutorial using Zenoh v0.5.0-beta9

From *peer-to-peer* to *brokered*

Changing the **communication model** in zenoh **does NOT require changing** the **application** logic nor the usage of a different API

It is enough to:

- Have at least one **zenoh-router** up and running in the system
- Set the zenoh **mode** to **“peer” / “client”** in the application

Note: Tutorial using Zenoh v0.5.0-beta9

Zenoh router

What is a zenoh router?

A zenoh router is a **software process** (i.e. **zenohd**) able to **route** the zenoh protocol between **clients** and **peers** in **any** given **topology**

A zenoh router also supports (*see next sections for details*):

- **Link-state protocol** for topology discovery and routing
- **Hybrid logic clock** for time stamping data as they are published and stored
- **Plugins** for integration with other technologies

Note: Tutorial using Zenoh v0.5.0-beta9

Install zenohd from Docker

Stable

```
$ docker pull eclipse/zenoh:latest
```

Nightly

```
$ docker pull eclipse/zenoh:master
```

Install zenohd from Eclipse

Stable

https://download.eclipse.org/zenoh/zenoh/latest/

Install zenohd from Deb

Stable

```
$ echo "deb [trusted=yes] https://
download.eclipse.org/zenoh/zenoh/latest/ /" |
sudo tee -a /etc/apt/sources.list > /dev/null
$ sudo apt update
$ sudo apt install zenoh
```

Install zenohd from source

Nightly

```
$ git clone https://github.com/eclipse-zenoh/
zenoh.git
$ cd zenoh
$ cargo build --release –all-targets
# The compiled zenohd can be found at:
# ./target/release/zenohd
```

Note: Tutorial using Zenoh v0.5.0-beta9

Start zenohd with default configuration

Usage

```
$ zenohd --help
```

Run

```
$ zenohd
```

Run with info log

```
$ RUST_LOG=info zenohd
```

Run with debug log

```
$ RUST_LOG=debug zenohd
```

Default network ports

TCP 7447 - For accepting new zenoh sessions

UDP 7447 - For scouting protocol

TCP 8000 - For REST API plugin (see later)

Docker CLI parameters

Expose/forward zenohd default ports

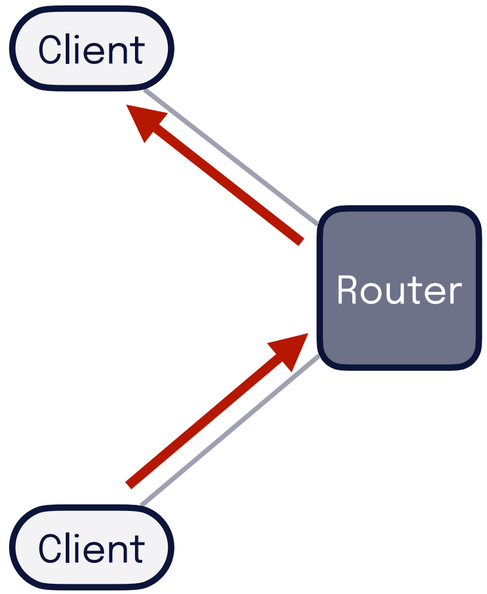
```
$ docker run --init -p 7447:7447/tcp \
  -p 7447:7447/udp -p 8000:8000/tcp  \
  eclipse/zenoh
```

Note: Tutorial using Zenoh v0.5.0-beta9

Pub/sub

/demo/example/**

Subscriber



Publisher

/demo/example/zenoh-python-pub

Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Publisher

```python
import time
import zenoh

# initiate logging
zenoh.init_logger()

# configure zenoh in client mode
conf = zenoh.Config.from_json5('{"mode": "client”}')

# open a zenoh session
session = zenoh.open(conf)

# publish
key_expr = "/demo/example/zenoh-python-pub"
val = "Hello!"
while True:
    time.sleep(1.0)
    session.put(key_expr, val)
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Subscriber

```python
import time
import zenoh

# initiate logging
zenoh.init_logger()

# configure zenoh in client mode
conf = zenoh.Config.from_json5('{"mode": "client"}')

# open a zenoh session
session = zenoh.open(conf)

# subscribe
def callback(sample):
    print("('{}': '{}')"
    .format(sample.key_expr, sample.payload))

key_expr = "/demo/example/**"
sub = session.subscribe(key_expr, callback)

while True:
    time.sleep(1.0)
```

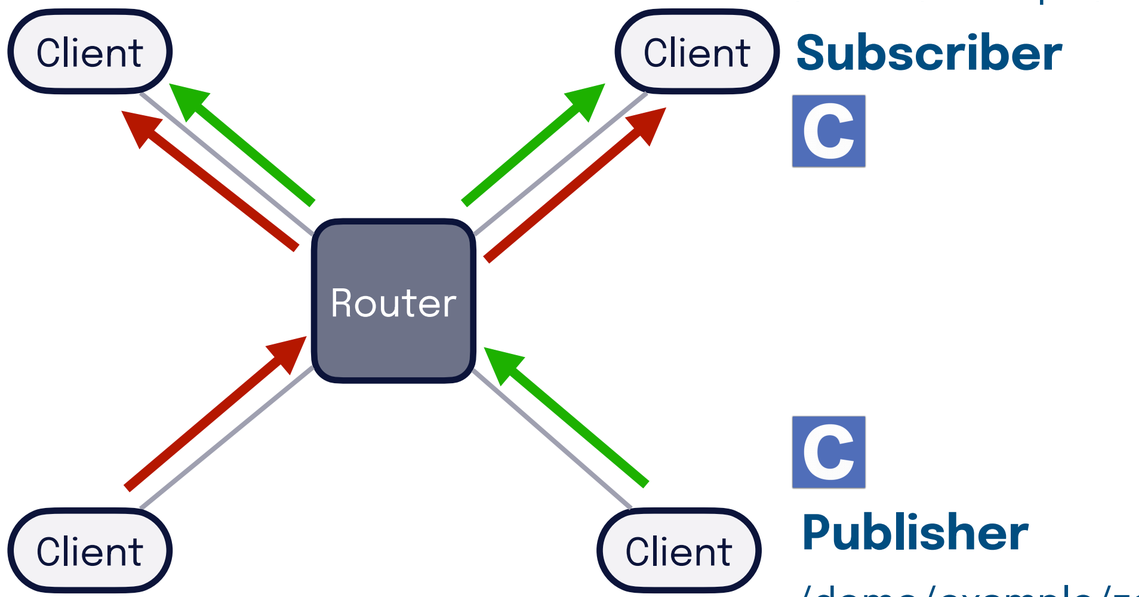
Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Note: Tutorial using Zenoh v0.5.0-beta9

Pub example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_pub.py
Sub example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_sub.py

/demo/example/**

Subscriber



/demo/example/**

Subscriber

Publisher

/demo/example/zenoh-python-pub

Publisher

/demo/example/zenoh-c-pub

Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Publisher

```c
#include <string.h>
#include <unistd.h>
#include <zenoh.h>

int main(int argc, char **argv) {
    // initiate logging
    z_init_logger();

    // open a zenoh session in client mode
    z_owned_config_t config = z_config_default();
    z_config_set(z_loan(config), ZN_CONFIG_MODE_KEY, "client");
    z_owned_session_t s = z_open(z_move(config));

    char buf[256];
    sprintf(buf, "%s", "Hello!");
    while (1) {
        // publish
        z_put(z_loan(s), z_expr("/demo/example/zenoh-c-pub")),
            (const uint8_t *)buf, strlen(buf));
        sleep(1);
    }

    return 0;
}
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Subscriber

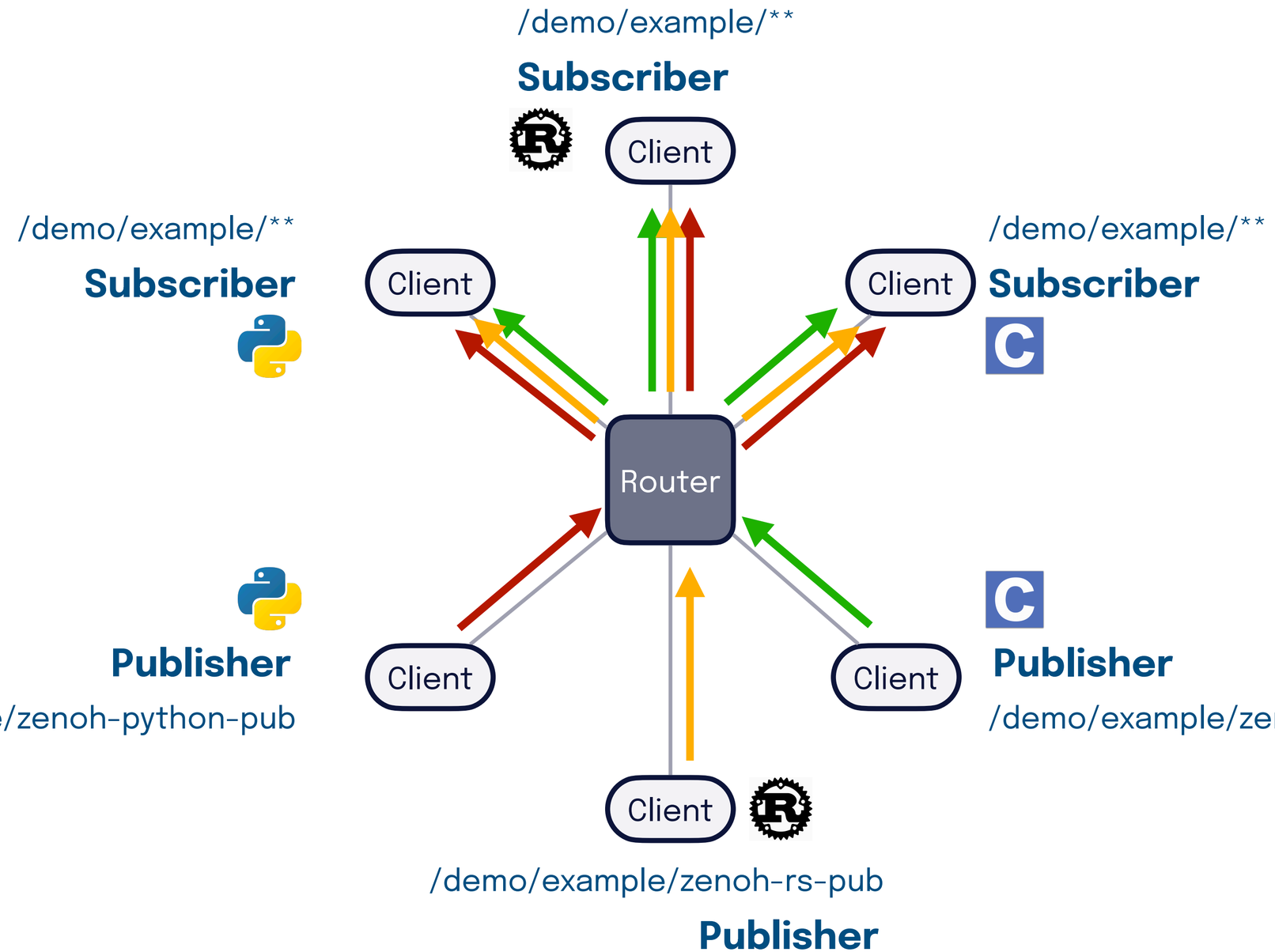
C

```c
#include <string.h>
#include <unistd.h>
#include <zenoh.h>

void callback(const z_sample_t *s, const void *arg) {
    printf("('%.*s': '%.*s')\n",
      (int)s->key.suffix.len, s->key.suffix.start,
      (int)s->value.len, s->value.start);
}

int main(int argc, char **argv) {
    // initiate logging
    z_init_logger();

    // open a zenoh session in client mode
    z_owned_config_t config = z_config_default();
    z_config_set(z_loan(config), ZN_CONFIG_MODE_KEY, "client");
    z_owned_session_t s = z_open(z_move(config));

    // subscribe
    z_owned_subscriber_t sub = z_subscribe(
         z_loan(s), z_expr("/demo/example/**"),
         z_subinfo_default(), callback, NULL);

    while (1) { sleep(1); } return 0;
}
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Note: Tutorial using Zenoh v0.5.0-beta9

Pub example: https://github.com/eclipse-zenoh/zenoh-c/blob/master/examples/z_pub.c
Sub example: https://github.com/eclipse-zenoh/zenoh-c/blob/master/examples/z_sub.c



Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Publisher

```rust
use async_std::task::sleep;
use std::time::Duration;
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;

#[async_std::main]
async fn main() {
    // initiate logging
    env_logger::init();

    // open a zenoh session in client mode
    let mut config = Config::default();
    config.set_mode(Some(WhatAmI::Client));
    let session = zenoh::open(config).await.unwrap();

    loop {
        // publish
        session.put("/demo/example/zenoh-rs-pub",
                    "Hello!").await.unwrap();
        sleep(Duration::from_secs(1)).await;
    }
}
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Subscriber

```rust
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;

#[async_std::main]
async fn main() {
    // initiate logging
    env_logger::init();

    // open a zenoh session in client mode
    let mut config = Config::default();
    config.set_mode(Some(WhatAmI::Client));
    let session = zenoh::open(config).await.unwrap();

    // subscribe
    let mut sub = session.subscribe("/demo/example/**")
                         .await.unwrap();

    while let Some(sample) = sub.receiver().next().await {
        println!("('{}': '{}')", sample.key_expr.as_str(),
            String::from_utf8_lossy(
                &sample.value.payload.contiguous()));
    }
}
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Note: Tutorial using Zenoh v0.5.0-beta9

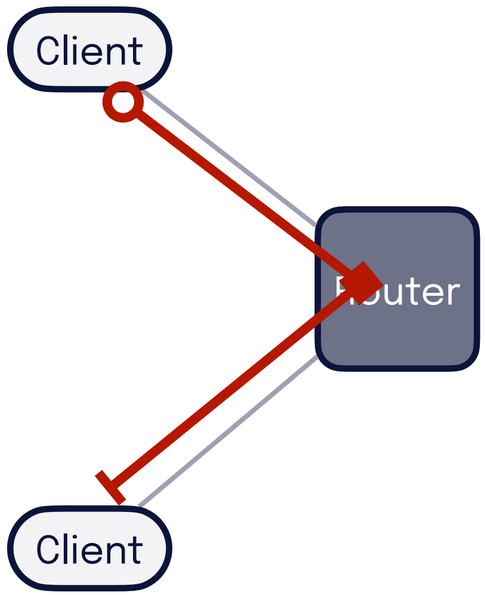
Pub example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_pub.rs
Sub example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_sub.rs

zetta scale

Distributed computed values

/demo/example/zenoh-python-eval

Eval



Get

/demo/example/**

Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Get

```python
import zenoh

# initiate logging
zenoh.init_logger()

# configure zenoh in client mode
conf = zenoh.Config.from_json5('{"mode": "client"}')

# open a zenoh session
session = zenoh.open(conf)

# query
replies = session.get_collect("/demo/example/**")
for r in replies:
    print("('{}': '{}')".format(
        reply.data.key_expr, reply.data.payload))
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

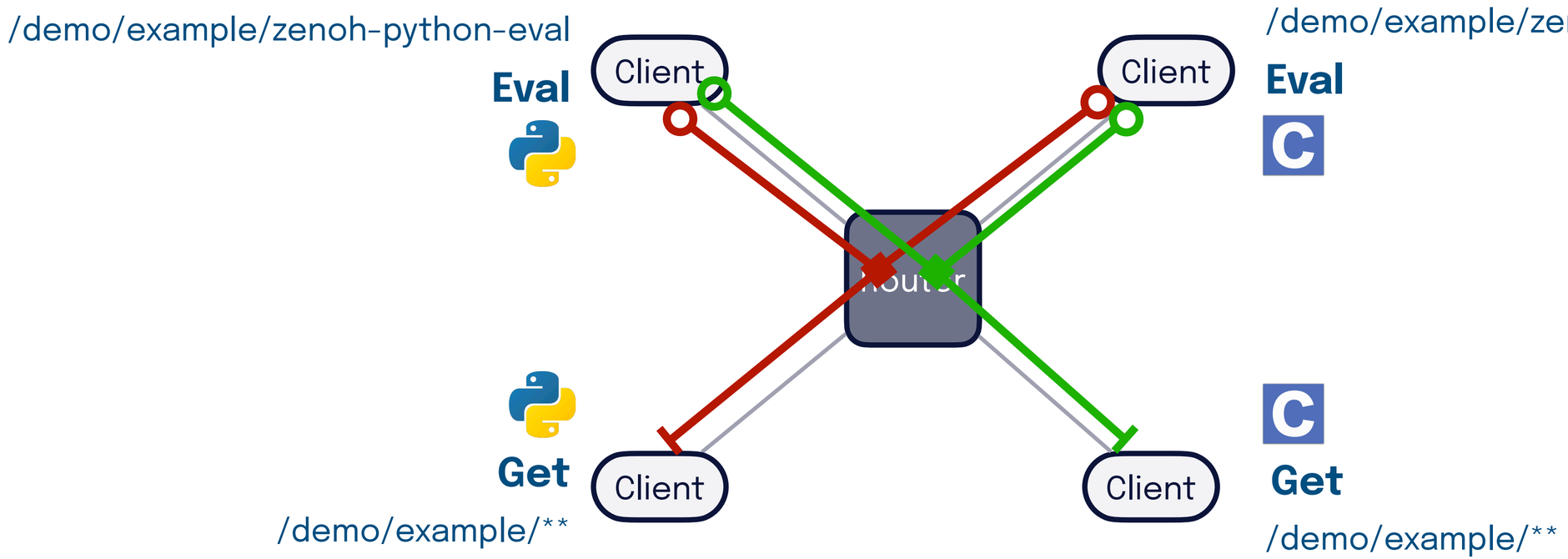
Eval

```python
import time
import zenoh
from zenoh import config, Sample
from zenoh.queryable import EVAL

# initiate logging
zenoh.init_logger()

# configure zenoh in client mode
conf = zenoh.Config.from_json5('{"mode": "client"}')

# open a zenoh session
session = zenoh.open(conf)

# eval
def callback(query):
    query.reply(Sample(key_expr=key,
        payload="Eval from Python!".encode()))

key_expr = "/demo/example/zenoh-python-eval"
queryable = session.queryable(key, EVAL, callback)

while True:
    time.sleep(1.0)
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Note: Tutorial using Zenoh v0.5.0-beta9

Get example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_get.py
Eval example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_eval.py



Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Get

```c
#include <zenoh.h>

int main(int argc, char **argv) {
    // initiate logging
    z_init_logger();
    // open a zenoh session in client mode
    z_owned_config_t config = z_config_default();
    z_config_set(z_loan(config), ZN_CONFIG_MODE_KEY, "client");
    z_owned_session_t s = z_open(z_move(config));
    // query
    z_query_target_t target = z_query_target_default();
    target.target.tag = z_target_t_ALL;

    z_owned_reply_data_array_t r = z_get_collect(z_loan(s),
           z_expr("/demo/example/**"), "", target,
           z_query_consolidation_default());

    for (unsigned int i = 0; i < r.len; ++i) {
        printf("('%.*s': '%.*s')\n",
            (int)r.val[i].data.key.suffix.len,
            r.val[i].data.key.suffix.start,
            (int)r.val[i].data.value.len,
            r.val[i].data.value.start);
    }

    return 0;
}
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

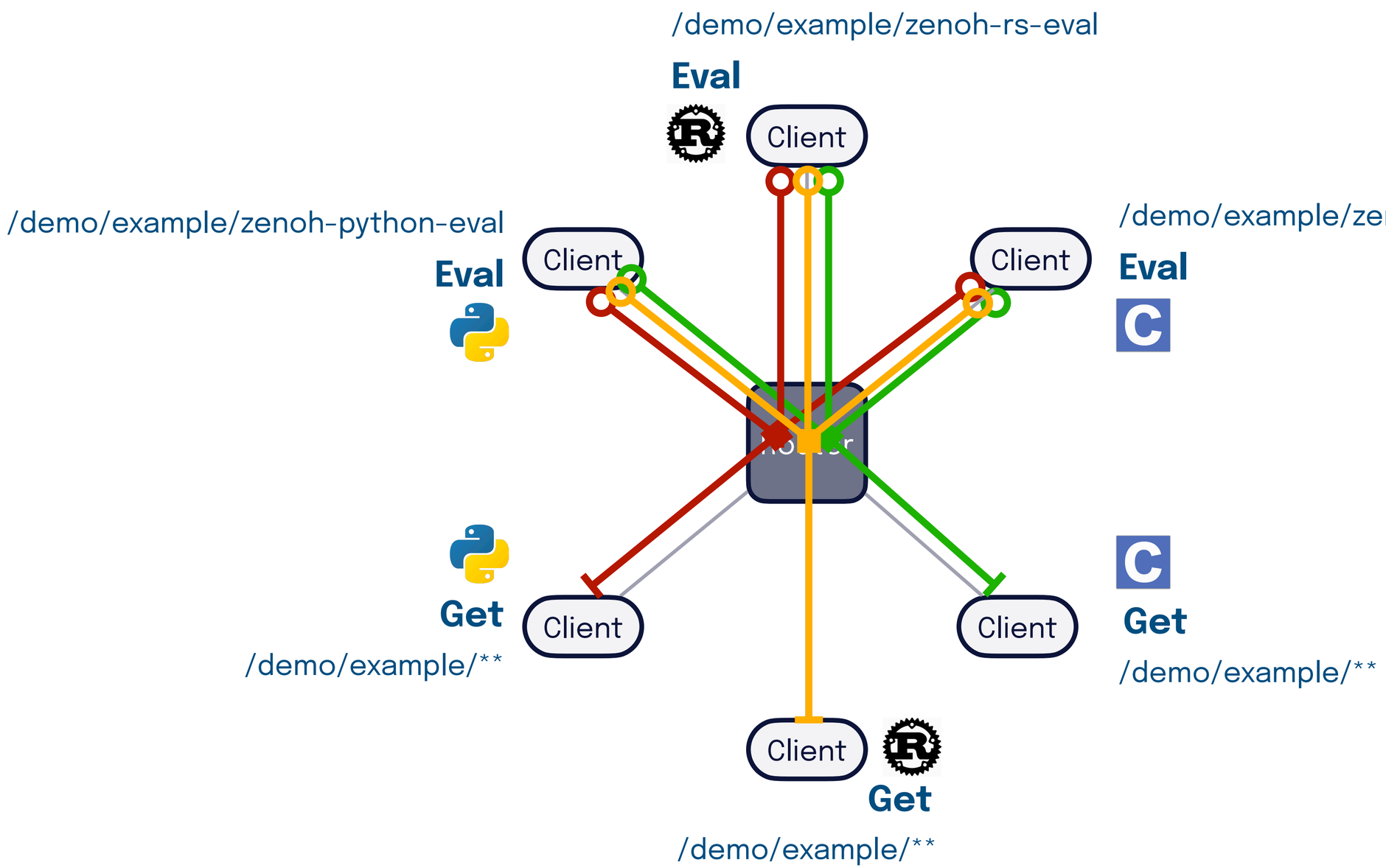
Eval

```c
#include <zenoh.h>

char *expr = "/demo/example/zenoh-c-eval";
char *value = "Eval from C!";

void callback(const z_query_t *query, const void *arg) {
    z_bytes_t res = z_query_key_expr(query).suffix;
    z_bytes_t pred = z_query_predicate(query);
    z_send_reply(query, expr,
        (const unsigned char *)value, strlen(value));
}

int main(int argc, char **argv) {
    // initiate logging
    z_init_logger();

    // open a zenoh session in client mode
    z_owned_config_t config = z_config_default();
    z_config_set(z_loan(config), ZN_CONFIG_MODE_KEY, "client");
    z_owned_session_t s = z_open(z_move(config));

    // eval
    z_owned_queryable_t qable = z_queryable_new(z_loan(s),
              z_expr(expr), ZN_QUERYABLE_EVAL, callback, NULL);

    while (1) { sleep(1); } return 0;
}
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Note: Tutorial using Zenoh v0.5.0-beta9

Get example: https://github.com/eclipse-zenoh/zenoh-c/blob/master/examples/z_get.c
Eval example: https://github.com/eclipse-zenoh/zenoh-c/blob/master/examples/z_eval.c



Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Get

```
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;

#[async_std::main]
async fn main() {
    // initiate logging
    env_logger::init();

    // open a zenoh session in client mode
    let mut config = Config::default();
    config.set_mode(Some(WhatAmI::Client));
    let s = zenoh::open(config).await.unwrap();

    // query
    let mut replies = s.get("/demo/example/**").await.unwrap();

    while let Some(reply) = replies.next().await {
        println!("('{}': '{}')",
           reply.data.key_expr.as_str(),
           String::from_utf8_lossy(&reply.data.value
               .payload.contiguous())
        );
    }
}
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

Eval

```
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;
use zenoh::queryable::EVAL;

#[async_std::main]
async fn main() {
    // initiate logging
    env_logger::init();

    // open a zenoh session in client mode
    let mut config = Config::default();
    config.set_mode(Some(WhatAmI::Client));
    let session = zenoh::open(config).await.unwrap();

    // eval
    let key_expr: &str = "/demo/example/zenoh-rs-eval";
    let mut qbl = session.queryable(key_expr)
                         .kind(EVAL).await.unwrap();

    while let Some(query) = qbl.receiver().next().await {
        query.reply(Sample::new(
            key_expr.clone(), value.clone()));
    }
}
```

Same application as in peer-to-peer!
Just set the mode to “client” or “peer”.

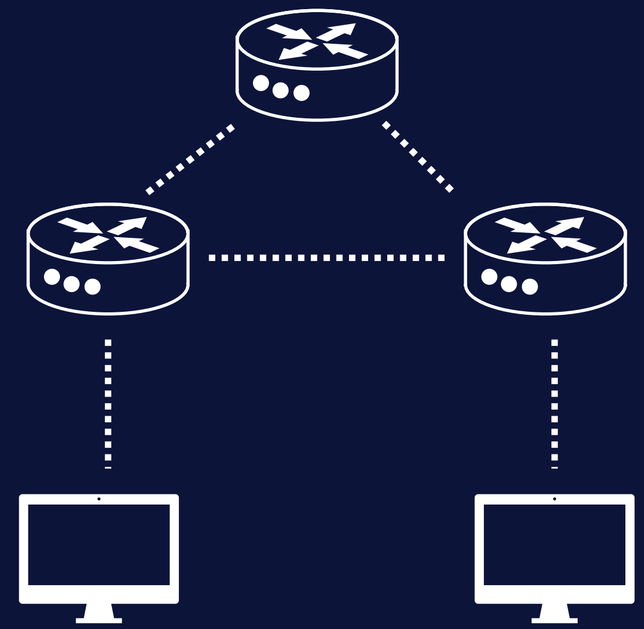
Note: Tutorial using Zenoh v0.5.0-beta9

Get example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_get.rs
Eval example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_eval.rs

zetta scale

Routed communication

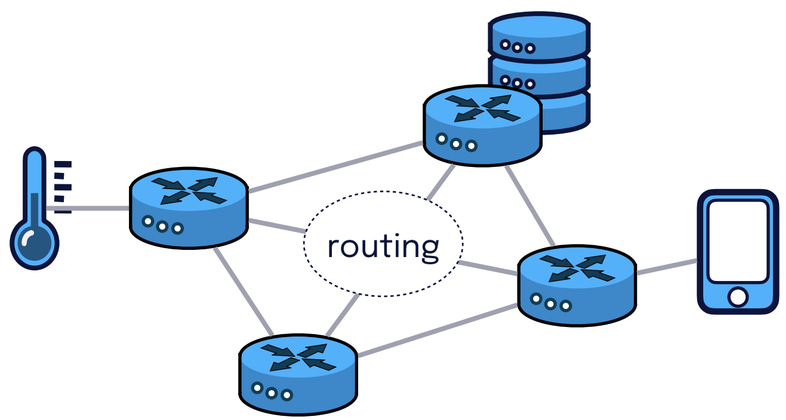
Let’s use many zenoh routers



What is *routed*?

Routed is a **communication model** where **multiple zenoh routers** cooperate with each other to **support** the communication between **peers** and **clients**

The zenoh routers use a **link state routing protocol** to automatically **forward data** to the right place in the zenoh infrastructure



Note: Tutorial using Zenoh v0.5.0-beta9

From *brokered* to *routed*

No changes to applications required:

- Configure the **“peer”** or **“client”** mode as in the *brokered* case

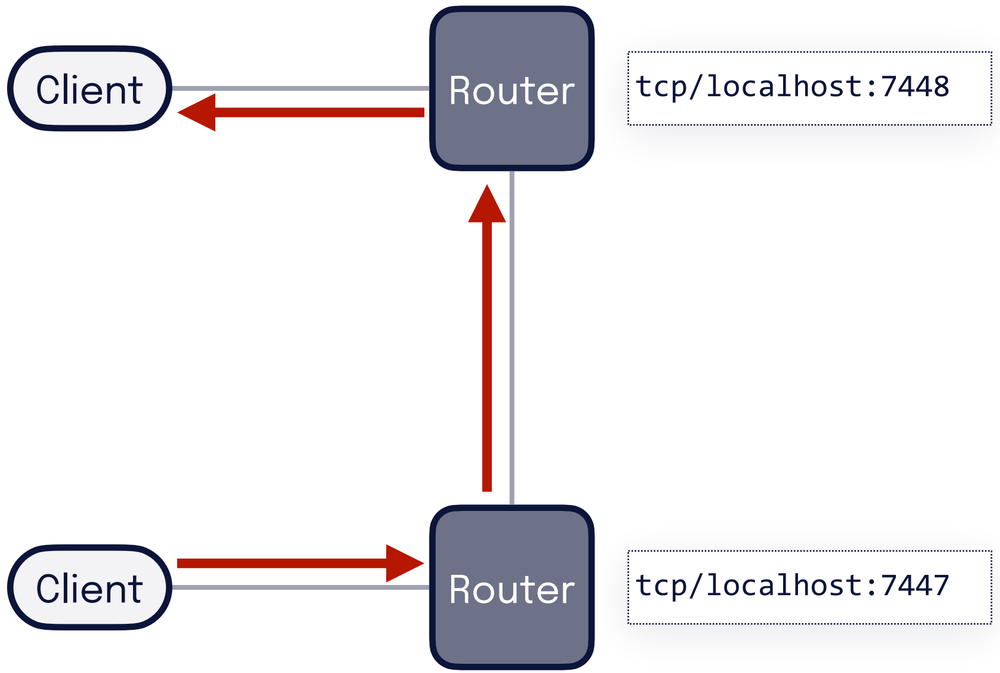
It is enough to:

- **Interconnect routers** between them via a proper configuration
- As many routers as needed in **any** required **topology**

Note: Tutorial using Zenoh v0.5.0-beta9

/demo/example/**

Subscriber



Publisher

/demo/example/zenoh-python-pub

Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Configure zenoh router interconnections

From CLI

```
$ zenohd --help
```

```
-l, --listen <ENDPOINT>...
    A locator on which this router will listen for
    incoming sessions. Repeat this option to open
    several listeners.

-e, --connect <ENDPOINT>...
    A peer locator this router will try to
    connect to. Repeat this option to connect to
    several peers.
```

```
(router 1)$ zenohd -l tcp/localhost:7447
```

```
(router 2)$ zenohd -l tcp/localhost:7448
  --rest-http-port 8001 -e tcp/localhost:7447
```

From configuration file

```
// conf1.json5
{
  mode: "router",
  listen: { endpoints: [ "tcp/localhost:7447" ] },
  plugins: { rest: { http_port: 8000, }, },
}
```

```
(router 1)$ zenohd -c conf1.json5
```

```
// conf2.json5
{
  mode: "router",
  listen: { endpoints: [ "tcp/localhost:7448" ] },
  connect: { endpoints: [ "tcp/localhost:7447" ] },
  plugins: { rest: { http_port: 8001, }, },
}
```

```
(router 2)$ zenohd -c conf2.json5
```

Note: Tutorial using Zenoh v0.5.0-beta9

Config example: https://github.com/eclipse-zenoh/zenoh/blob/master/EXAMPLE_CONFIG.json5

Connect applications to specific zenoh routers

Python

```
conf = zenoh.Config.from_json5({
  "mode": “client" // Or "peer"
  "connect": { "endpoints": [ "tcp/localhost:7447" ] },
  "scouting": { "multicast": { "enabled": false } }
})
```

Rust

```
let mut conf = Config::default();
config.set_mode(Some(WhatAmI::Client));
                  // Or "WhatAmI::Peer"
config.connect.endpoints.push(
    "tcp/localhost:7447".parse().unwrap());
config.scouting.multicast.set_enabled(Some(false));
```

You can also disable multicast scouting

C

```
z_owned_config_t config = z_config_default();
z_config_set(z_loan(config),
    ZN_CONFIG_MODE_KEY, "client"); // Or "peer"
z_config_set(z_loan(config),
   ZN_CONFIG_CONNECT_KEY, "tcp/localhost:7447");
```

Configuration file

```
{
  mode: "client", // Or "peer"
  connect: { endpoints: [ "tcp/localhost:7447" ] },
  scouting: { multicast: { enabled: false } }
}
```

No more changes to the code with a config file!

Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

Zenoh best practices

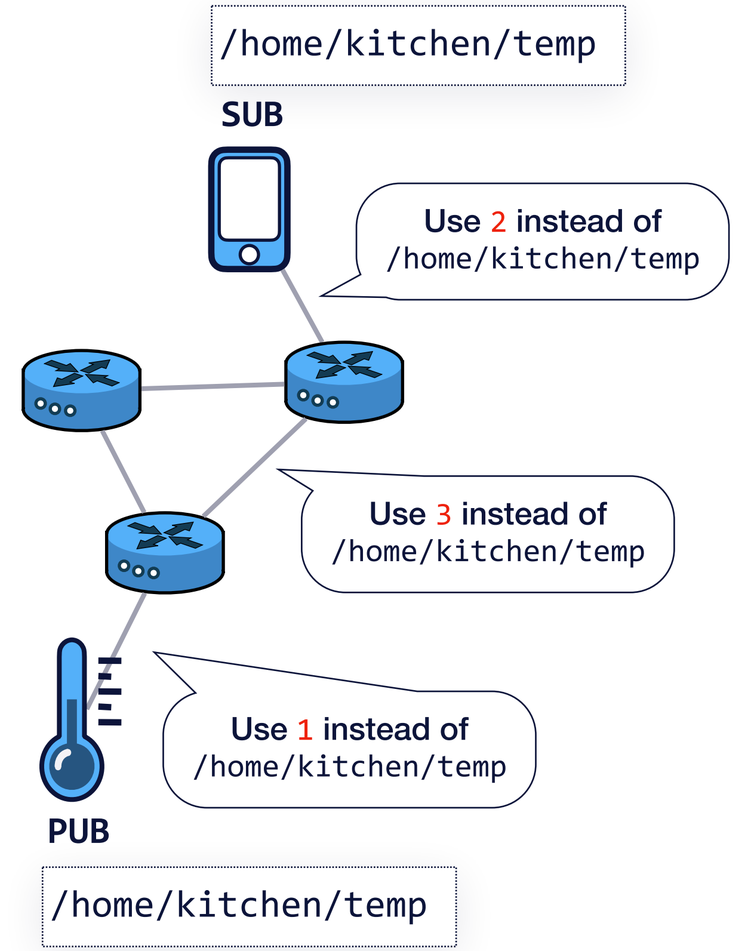
Few lines for a great benefit

Resource declaration

Declaring a **resource maps long resource keys** expressed in string-form **to small integers** on the wire

Using integers as resource keys **reduces** wire **overhead**, **saves bandwidth**, and **increases throughput**

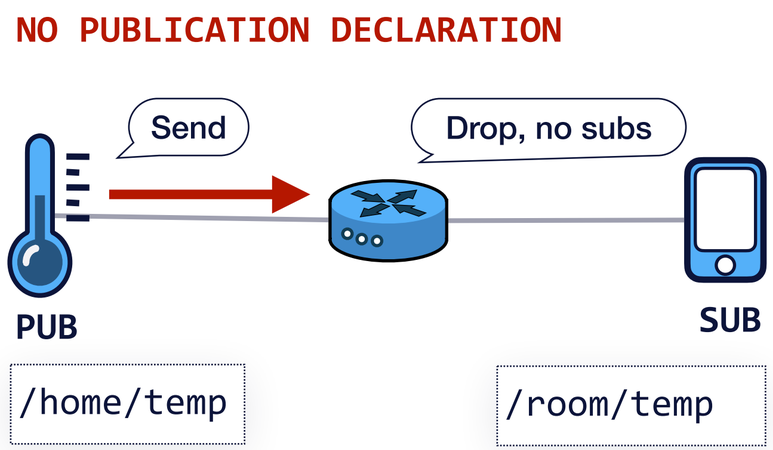
The **resource mapping** is **local** to each hop and no global synchronisation is required in the whole system

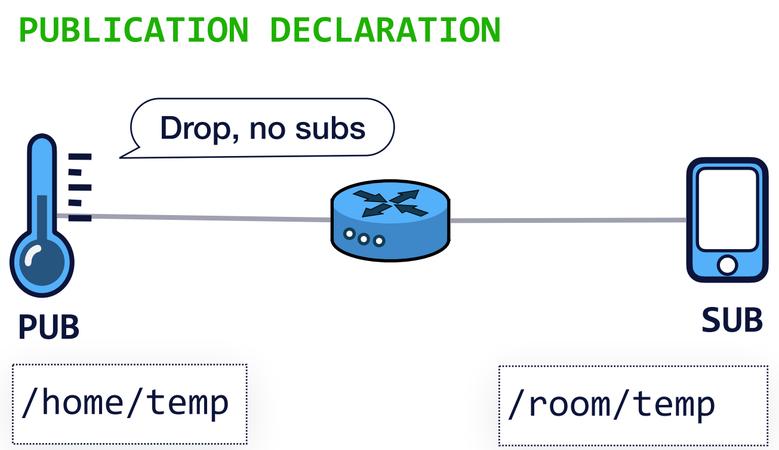


Note: Tutorial using Zenoh v0.5.0-beta9

Publication declaration

Declaring a **publication** allows to **not send data** on the wire when there are **no matching subscribers**, thus **saving bandwidth** from unnecessary transmissions





Note: Tutorial using Zenoh v0.5.0-beta9

Publisher

```
import time
import zenoh

# initiate logging
zenoh.init_logger()

# configure zenoh in peer mode
conf = zenoh.Config.from_json5('{"mode": "client"}')

# open a zenoh session
session = zenoh.open(conf)

# publish
key_expr = "/demo/example/zenoh-python-pub"
val = "Hello!"

# declarations
rid = session.declare_expr(key_expr) # first declare the resource
session.declare_publication(rid)   # then declare the publication

while True:
    time.sleep(1.0)
    # use rid (integer) instead of key_expr (string)
    session.put(rid, val)
```

Subscriber

```
import time
import zenoh

# initiate logging
zenoh.init_logger()

# configure zenoh in peer mode
conf = zenoh.Config.from_json5('{"mode": "client"}')

# open a zenoh session
session = zenoh.open(conf)

# subscribe
def callback(sample):
    print("('{}': '{}')"
    .format(sample.key_expr, sample.payload))

key_expr = "/demo/example/**"
# declaration
rid = session.declare_expr(key_expr)
# use rid (integer) instead of key_expr (string)
sub = session.subscribe(rid, callback)

while True:
    time.sleep(1.0)
```

Note: Tutorial using Zenoh v0.5.0-beta9

Pub example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_pub.py
Sub example: https://github.com/eclipse-zenoh/zenoh-python/blob/master/examples/z_sub.py

zetta scale

REST API

A zenoh-router plugin

A web-friendly API

Provided by the zenoh router via the **REST plugin**

```
Default TCP port: 8000
```

It allows to:

- Map the zenoh **put/get/delete** operations to the **PUT/GET/DELETE** HTTP methods
- Returns values in **JSON** format

Note: Tutorial using Zenoh v0.5.0-beta9

Using *cURL* for the REST API

```
# Put a string value in /demo/example/zenoh-rest-pub
curl -X PUT -d 'Hello World!' 'http://localhost:8000/demo/example/zenoh-rest-pub'

# Put a JSON value in /demo/example/zenoh-rest-json
curl -X PUT -H "content-Type: application/json" -d '{"value": "Hello World!"}' \
'http://localhost:8000/demo/example/zenoh-rest-json'

# Get the keys/values matching /demo/example/**
curl -X GET 'http://localhost:8000/demo/example/**'

# Delete key/value /demo/example/zenoh-rest-pub
curl -X DELETE 'http://localhost:8000/demo/example/zenoh-rest-pub'
```

Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

Admin space

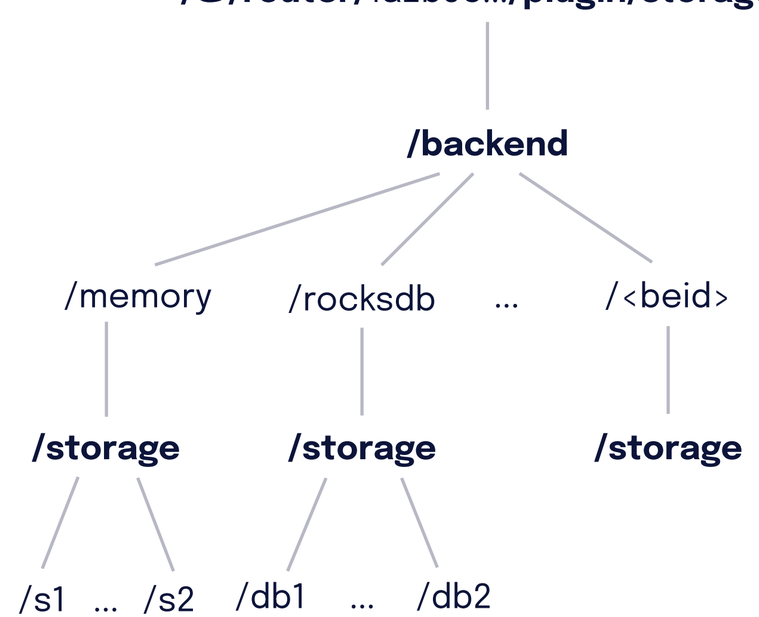
Managing the zenoh infrastructure

Dedicated resources for admin

A zenoh key/value space with under **/@/*****

Addressable via any zenoh API using **put/get/delete** operations (e.g., python, C, Rust, REST, etc.)

Each router addressable via its “**ID**” or the "**local**" keyword for the router an API is connected to



Note: Tutorial using Zenoh v0.5.0-beta9

Querying the admin space

We can use *cURL* and the REST API

```
# Get local router's info
curl http://localhost:8000/@/router/local

# Get all routers' of the system
curl http://localhost:8000/@/router/*

# Get local router's storages info
curl http://localhost:8000/@/router/local/**/storages/**
```

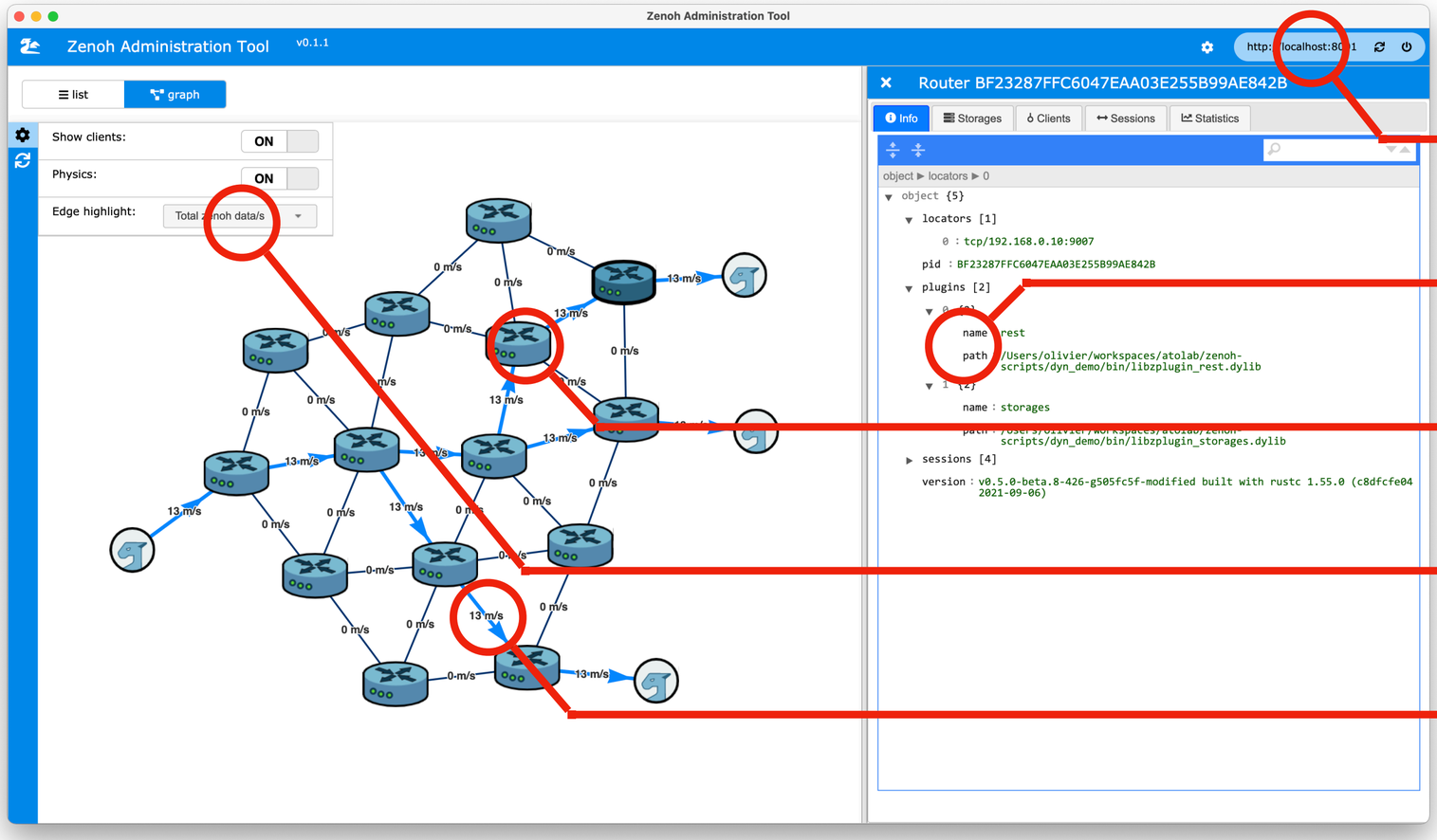
Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

zenoh administration tool

zenoh administration tool

Available in the commercial version



Connect to a zenoh router

Inspect router status

Discover system topology

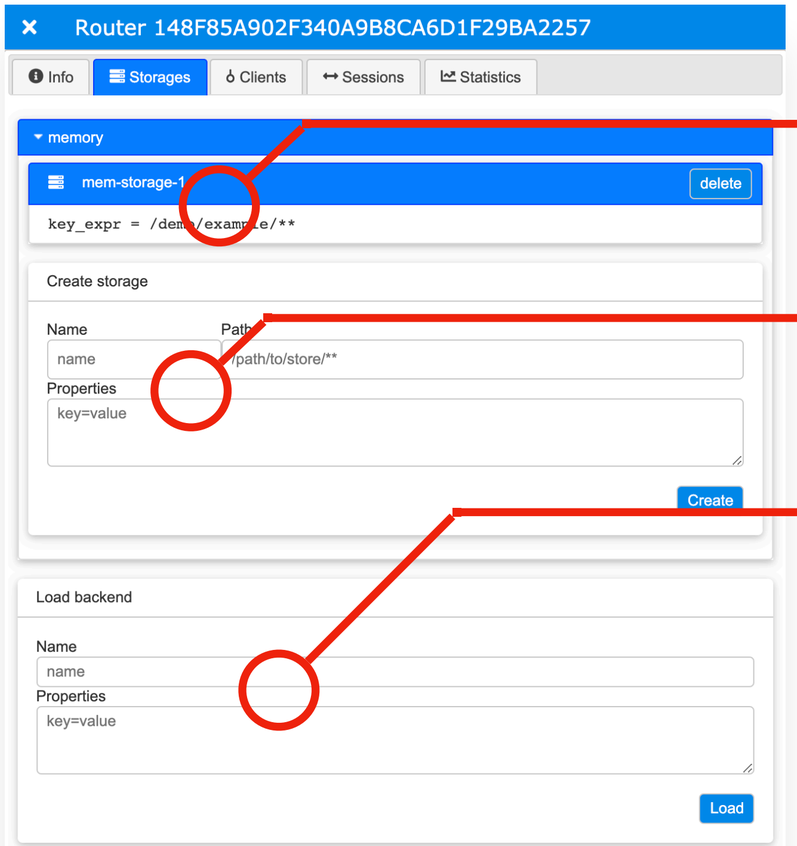
Configure traffic highlight

Visualise traffic

Note: Tutorial using Zenoh v0.5.0-beta9

zenoh administration tool

Available in the commercial version

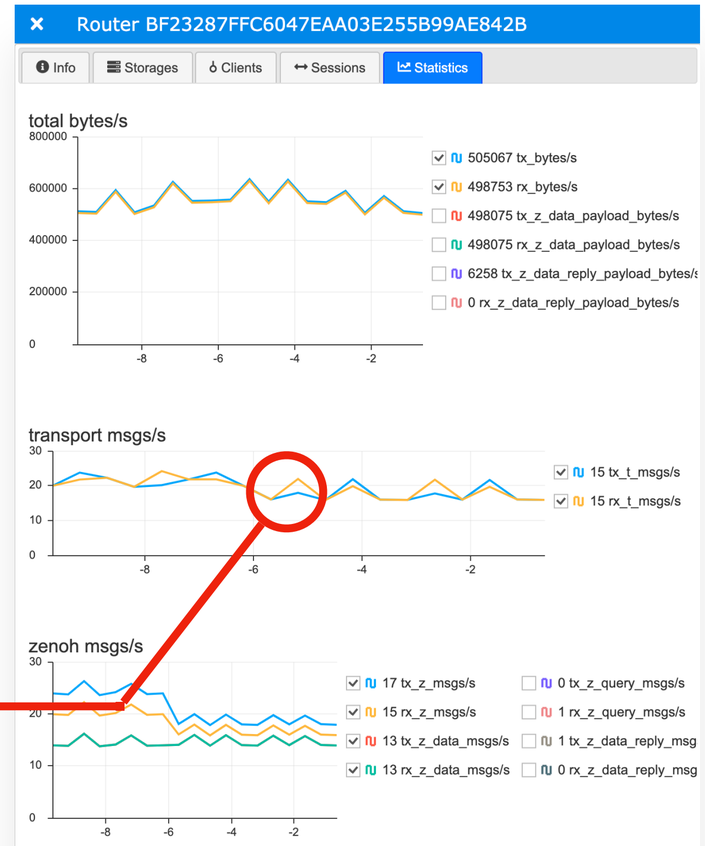


Inspect storages

Create storages

Load backends

Display router statistics



Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

Plugins and backends

Enrich the zenoh-router with more functionalities

Plugin

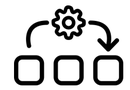
It allows to **integrate** zenoh with **different technologies** (e.g. storage) via a common interface

It is a **dynamic library** that can be loaded at start-up time by the **zenoh router**

It takes the name of `libzplugin_<name>.so`

https://github.com/eclipse-zenoh/zenoh/tree/master/plugins/zenoh-plugin-trait

Backend



It provides a **targeted implementation** of a given plugin (e.g. rocksdb) via a common interface

It is a **dynamic library** that can be loaded at start-up time by the **zenoh router**

It takes the name of `libzbackend_<name>.so`

https://github.com/eclipse-zenoh/zenoh/tree/master/plugins/zenoh-backend-traits

Note: Tutorial using Zenoh v0.5.0-beta9

Available plugins

REST

{ REST }

- Maps zenoh primitives to HTTP methods
- It support HTML5 Server Sent Events
- Built-in zenoh router - Enabled by default

DDS

- Carries DDS traffic on top of zenoh
- Transparent to DDS participants

https://github.com/eclipse-zenoh/zenoh-plugin-dds

Storage

- A geo-distributed key-value store
- Alignment of decentralised storages
- Built-in zenoh router

Webserver

- Serves web content (e.g., pages) that is stored in zenoh decentralised storages

https://github.com/eclipse-zenoh/zenoh-plugin-webserver

Note: Tutorial using Zenoh v0.5.0-beta9

Storages

Storage backends

In-memory

- A volatile key-value store
- Useful for last-value fast caching
- Built-in in the zenoh router

RocksDB

- A persistent key-value store
- Useful for last-value persistence

https://github.com/eclipse-zenoh/zenoh-backend-rocksdb

InfluxDB

- A persistent database for time series
- Useful for performing data analytics

https://github.com/eclipse-zenoh/zenoh-backend-influxdb

Filesystem

- A persistent key-value store on filesystem
- Useful for serving large files (e.g., CDN)

https://github.com/eclipse-zenoh/zenoh-backend-filesystem

Note: Tutorial using Zenoh v0.5.0-beta9

Storage backend configuration examples

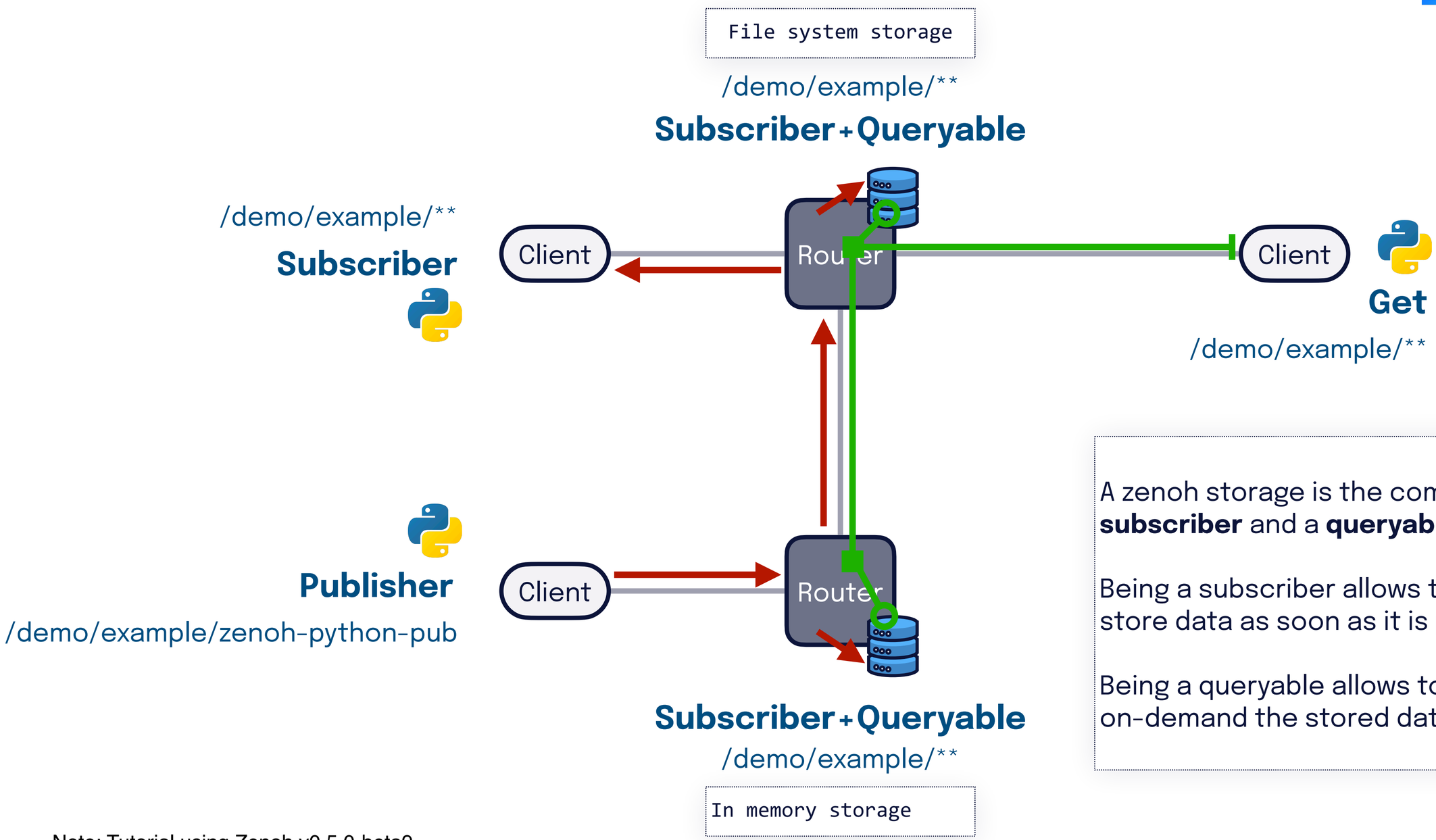
In-memory

```
{
 plugins: {
  storages: {
   backends: {
    memory: {
     storages: {
      // configuration of a "demo" storage
      // using the "memory" backend
      demo: {
       // the key expression this
       // storage will subscribes to
       key_expr: "/demo/example/**",
       // this prefix will be stripped from
       // the received key when converting
       // to file path
       strip_prefix: "/demo/example",
} } } } } } }
```

Filesystem

```
{
 plugins: {
  storages: {
   backends: {
    fs: {
     storages: {
      // configuration of a "demo" storage
      // using the "fs" backend
      demo: {
       // the key expression this
       // storage will subscribes to
       key_expr: "/demo/example/**",
       // this prefix will be stripped from
       // the received key when converting
       // to file path
       strip_prefix: "/demo/example",
        // the key/values will be stored as
        // files within this directory
        dir: "example"
} } } } } } }
```

Note: Tutorial using Zenoh v0.5.0-beta9



A zenoh storage is the combination of a **subscriber** and a **queryable**

Being a subscriber allows to **receive** and store data as soon as it is published

Being a queryable allows to **retrieve** on-demand the stored data

Note: Tutorial using Zenoh v0.5.0-beta9

Make sure zenohd is up and running

zetta scale

Query tuning

Tuning

In some cases it is desirable to tune the query in order to:

- Reduce the **latency** in receiving the replies
- Reduce the **state** kept on the router
- Save **bandwidth** on constrained links

Query tuning operates on:

- **Targeting**
- **Consolidation**

Targeting

Indicates the target of the query based on:

kind (e.g., eval, storage)
proximity (e.g., best-matching, all)
completeness (e.g., complete only)

Consolidation

How and **where** replies are **filtered**

It allows to choose the right **tradeoff** between **bandwidth, latency** and **state** in the routers

Note: Tutorial using Zenoh v0.5.0-beta9

Query targeting

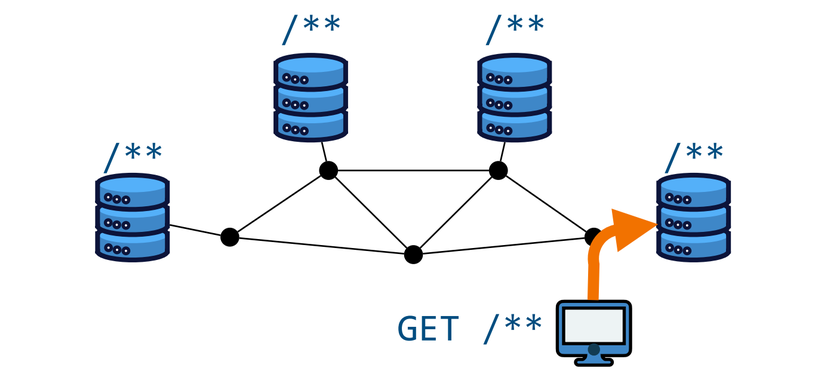
Completeness

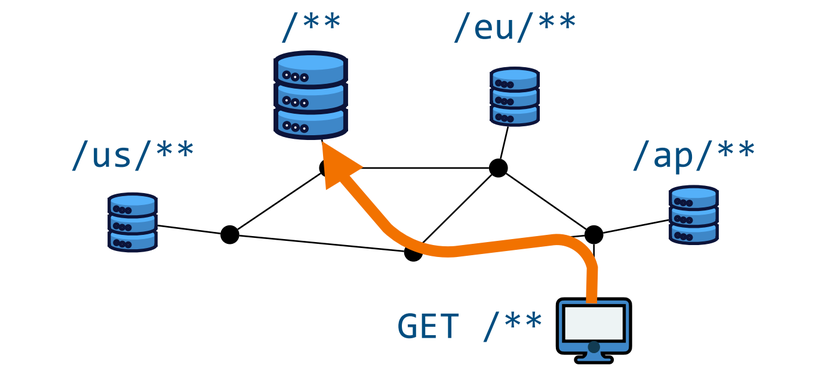
A storage is considered **complete** with respect to a query if it eventually contains **all the resources** matching the **query expression**.

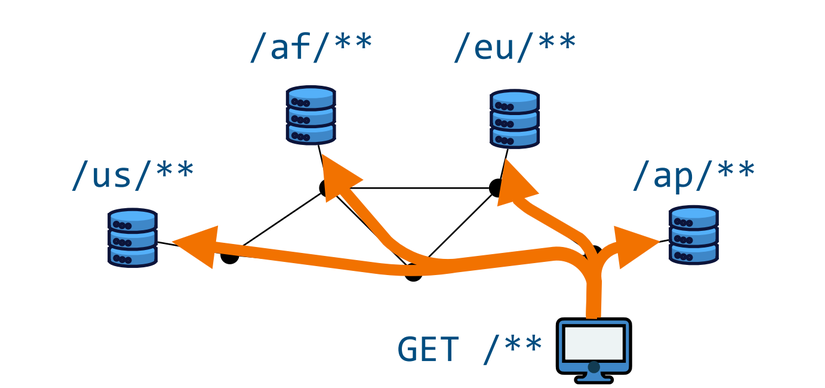
Note: when declaring a storage or a queryable you may **explicitly** declare it **incomplete**.

Meaning that it does not contain all the resources matching its expression.

Note: Tutorial using Zenoh v0.5.0-beta9







BEST-MATCHING

Get data from the **nearest complete** storage if any or all if none

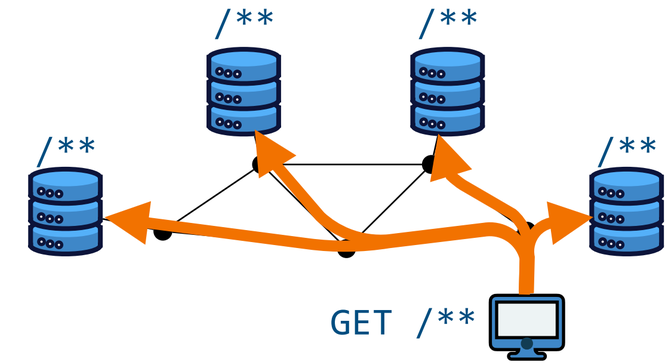
ALL

Get data from all matching storages

ALL_COMPLETE

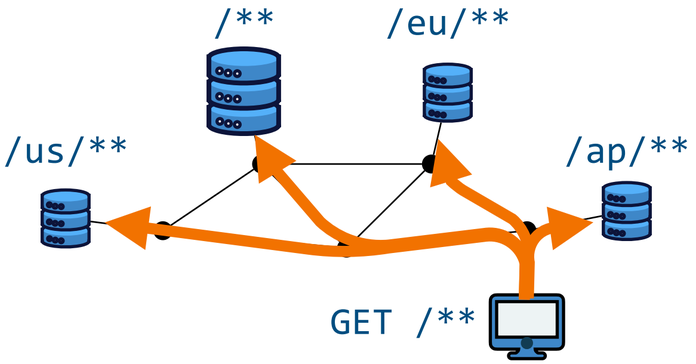
Get data from complete storages only

Note: Tutorial using Zenoh v0.5.0-beta9



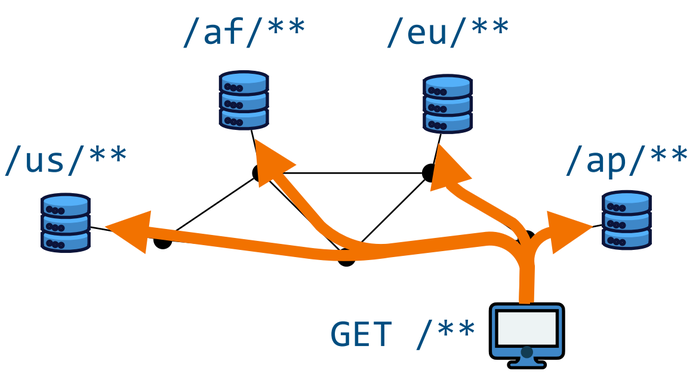
BEST-MATCHING

Get data from the nearest complete storage if any or all if none



ALL

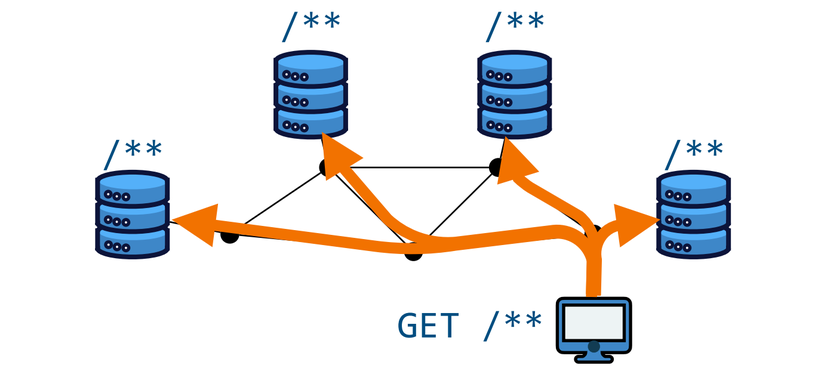
Get data from **all** matching storages

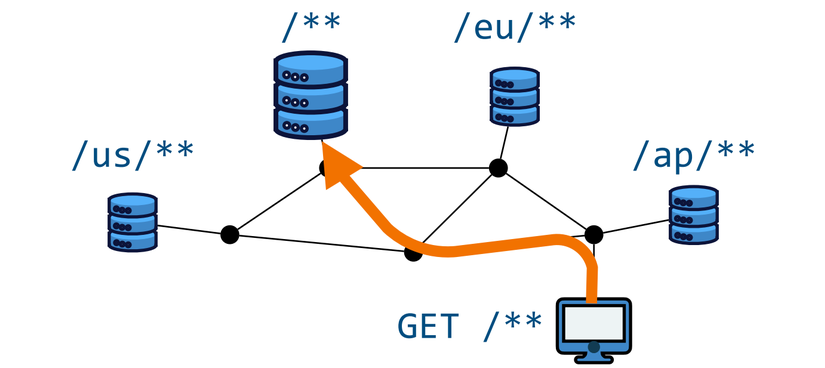


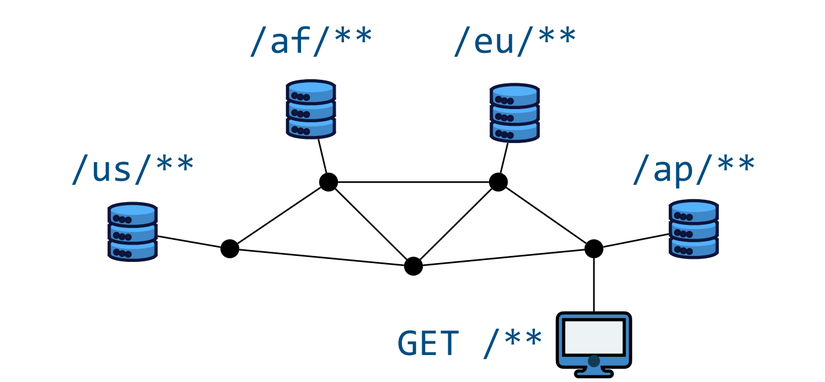
ALL_COMPLETE

Get data from complete storages only

Note: Tutorial using Zenoh v0.5.0-beta9







BEST-MATCHING

Get data from the nearest complete storage if any or all if none

ALL

Get data from all matching storages

ALL_COMPLETE

Get data from **complete** storages only

Note: Tutorial using Zenoh v0.5.0-beta9

Query consolidation

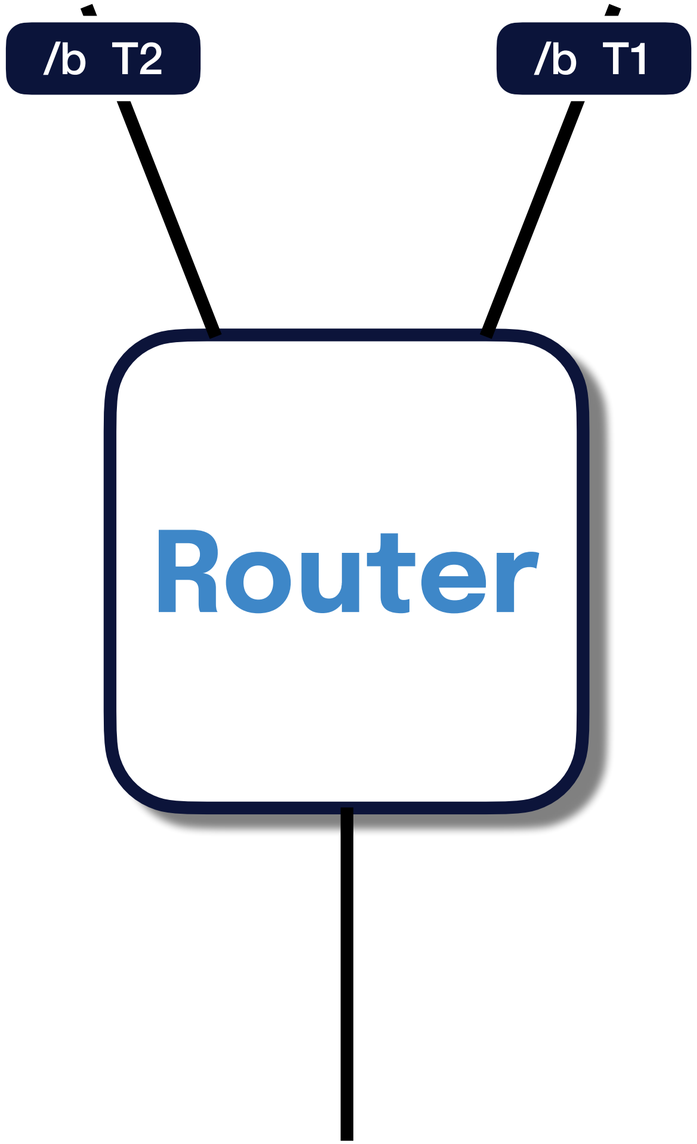
Query consolidation

Filter out multiple **values** for the same **key** to only keep the most **recent** one

Consolidation is performed in **each router** along the **route** traversed by the replies in order to save **bandwidth**

Note: Tutorial using Zenoh v0.5.0-beta9

Consolidation modes



NONE

All replies produced by the matching queryables are sent back to the querier

This mode is useful when dealing with time series

LAZY

Only replies more recent than the latest seen are forwarded back to the querier

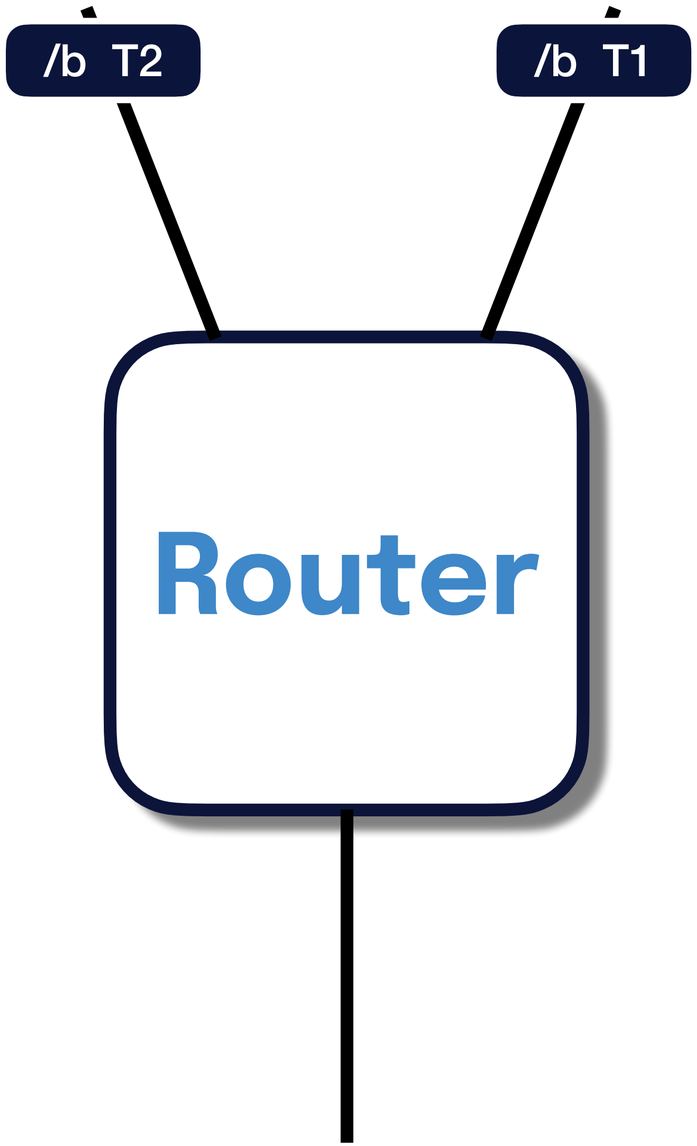
It allows to save bandwidth while requiring a minimal state to be kept in the zenoh routers

FULL

All the replies are first collected and then only the most recent one is forwarded back

It is the most demanding consolidation mode in terms of state. It usually done only in the API.

Consolidation modes



NONE

All replies produced by the matching queryables are sent back to the querier

This mode is useful when dealing with time series

LAZY

Only replies more recent than the latest received are forwarded back to the querier

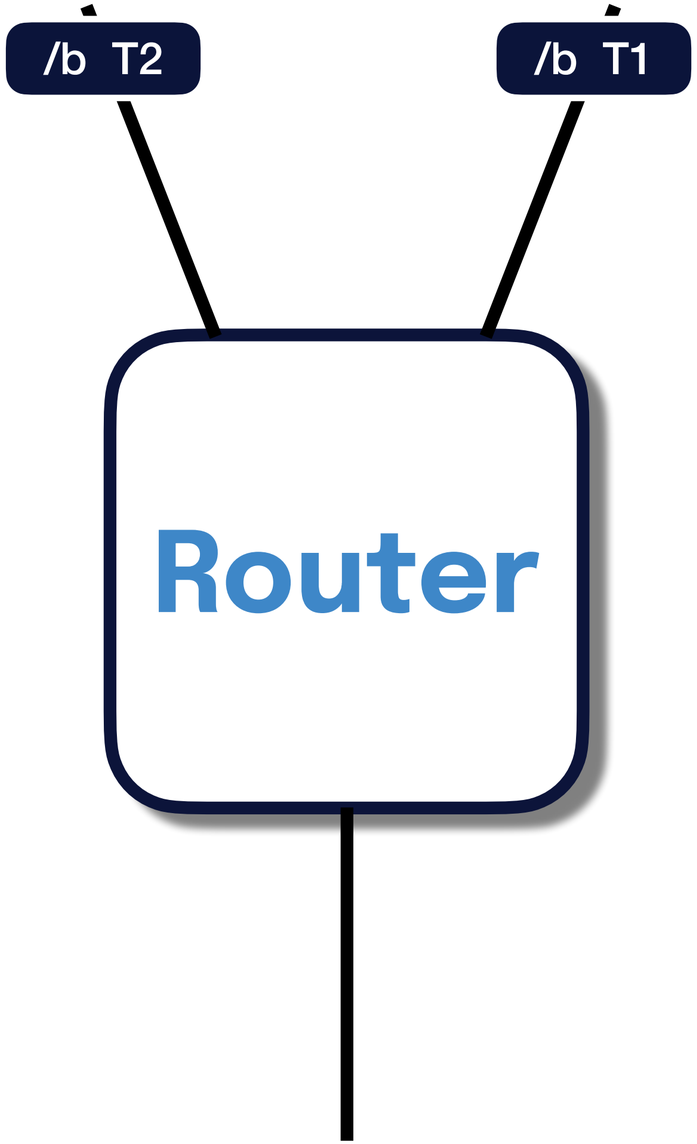
It allows to save bandwidth while requiring a minimal state to be kept in the zenoh routers

FULL

All the replies are first collected and then only the most recent one is forwarded back

It is the most demanding consolidation mode in terms of state. It usually done only in the API.

Consolidation modes



NONE

All replies produced by the matching queryables are sent back to the querier

This mode is useful when dealing with time series

LAZY

Only replies more recent than the latest seen are forwarded back to the querier

It allows to save bandwidth while requiring a minimal state to be kept in the zenoh routers

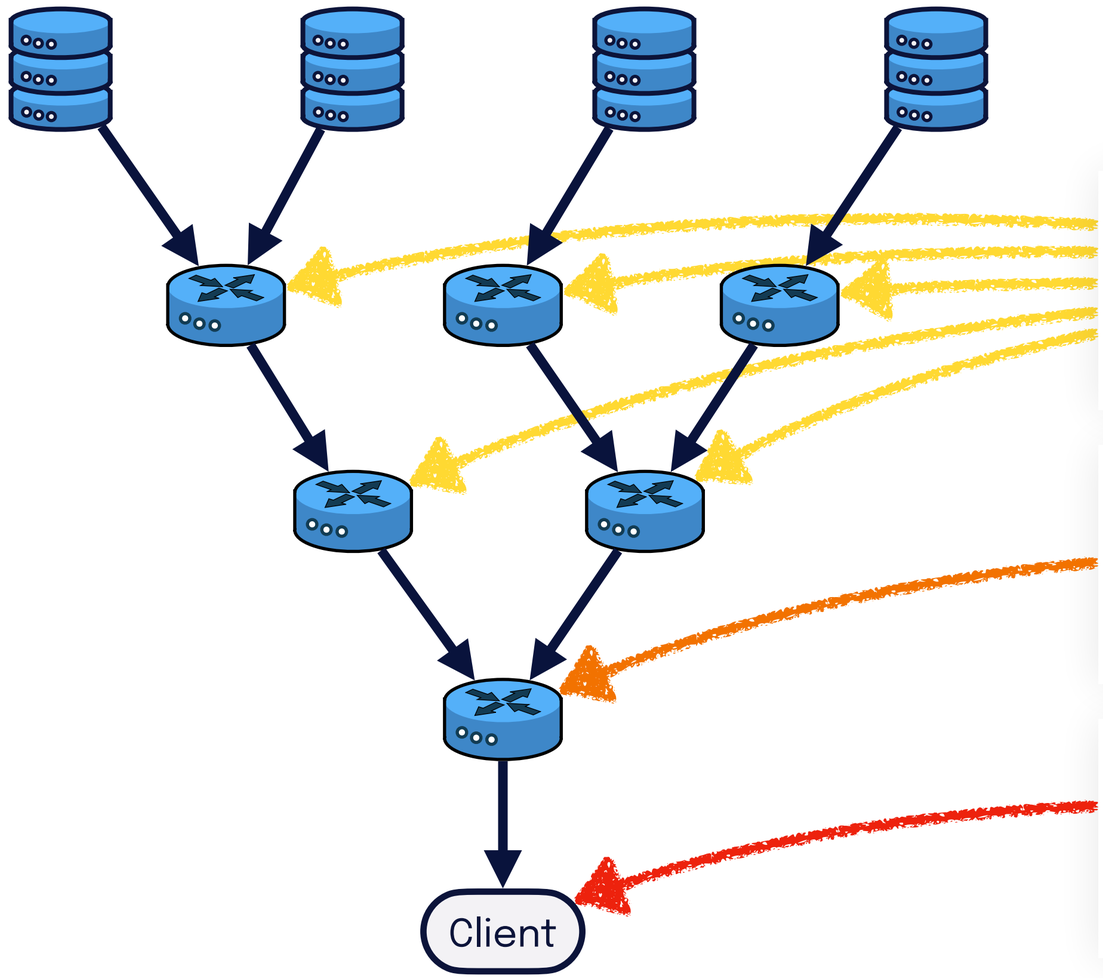
FULL

All the replies are first collected and then only the most recent one is forwarded back

It is the most demanding consolidation mode in terms of state.

It is usually done only in the API.

Query consolidation



first_routers

Default: **Lazy**

The consolidation mode to apply on all routers except the last one in the replies route

last_router

Default: **Lazy**

The consolidation mode to apply on the last router of the replies route

reception

Default: **Full**

The consolidation mode to apply in the final client

Note: Tutorial using Zenoh v0.5.0-beta9

Get

```
use zenoh::config::Config;
use zenoh::prelude::config::WhatAmI;
use zenoh::query::*;

#[async_std::main]
async fn main() {
    // open zenoh session…

    // query
    let query_target = QueryTarget {
        kind: zenoh::queryable::STORAGE,
        target: Target::AllComplete,
    };
    let query_consolidation = QueryConsolidation {
        first_routers: ConsolidationMode::Full,
        last_router: ConsolidationMode::Full,
        reception: ConsolidationMode::Full
    };
    let mut replies = session.get(&selector)
        .target(query_target)
        .consolidation(query_consolidation)
        .await.unwrap();

    while let Some(reply) = replies.next().await
    { // handle replies… }
}
```

Query parameters

Selector: a filter for the set of resources

Kind: the kind of queryables the query is interested in.
Valid values are:

```
['ALL_KINDS', 'STORAGE', 'EVAL']
```

Target: the set of queryables targeted by the query.
Valid values are:

```
['ALL', 'BEST_MATCHING', 'ALL_COMPLETE', 'NONE']
```

Consolidation: the consolidation applied to the replies of the query. Valid values are:

```
['NONE', 'LAZY', 'FULL']
```

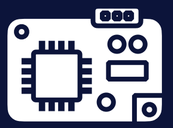
Note: Tutorial using Zenoh v0.5.0-beta9

Get example: https://github.com/eclipse-zenoh/zenoh/blob/master/examples/examples/z_get.rs

zetta scale

Zenoh-pico

For embedded devices and microcontrollers



What is *Zenoh-Pico*?

Small footprint **zenoh library** optimised for **embedded systems and microcontrollers**

Natively implemented in C

(~80% of embedded development)

Can be easily **ported** to many **boards** and **platforms**

Note: Tutorial using Zenoh v0.5.0-beta9

Prepare your environment

PlatformIO can ease the building and uploading process

Website: https://platformio.org/

Search for your board

```
$ platformio boards
```

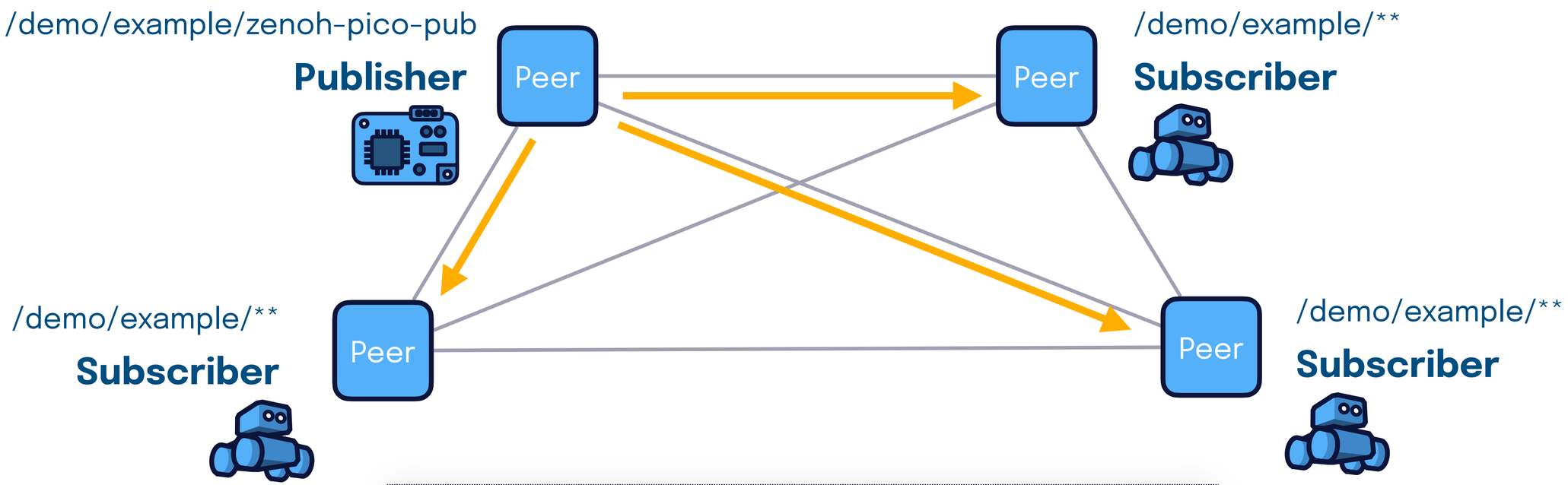
Create your project

```
$ platformio init -b <YOUR BOARD ID>
```

Note: Tutorial using Zenoh v0.5.0-beta9

Pub/sub

Peer-to-peer communication




```
At the moment, peer-to-peer communication in zenoh-pico is
only available in case of UDP over IP multicast.

Full peer-to-peer support as in zenoh-c is under development.
```

Note: Tutorial using Zenoh v0.5.0-beta9

Publisher

```
#include <Arduino.h>
#include <WiFi.h>

extern "C"
{
    #include "zenoh-pico.h"
}

zn_session_t *s = NULL;
zn_reskey_t *reskey = NULL;

void setup()
{
    WiFi.mode(WIFI_STA);
    WiFi.begin(“MY_SSID", “MY_PASS");
    while (WiFi.status() != WL_CONNECTED)
        delay(1000);
```

Continue… =>

C

```
zn_properties_t *config = zn_config_default();
zn_properties_insert(config, ZN_CONFIG_MODE_KEY,
    z_string_make(“peer”));
zn_properties_insert(config, ZN_CONFIG_PEER_KEY,
    z_string_make(“udp/224.0.0.225:7447#iface=eth0”));

s = zn_open(config);
if (s == NULL)
    return;

    znp_start_read_task(s);
    znp_start_lease_task(s);

    *reskey = zn_rname(“/demo/example/zenoh-pico-pub”);
}

void loop()
{
    delay(5000);

    char *buf = "Publishing data from ESP32";
    zn_write(s, *reskey, (const uint8_t *)buf, strlen(buf));
}
```

Note: Tutorial using Zenoh v0.5.0-beta9

Pub example: https://github.com/eclipse-zenoh/zenoh-pico/blob/master/examples/net/esp32/zn_pub.ino

Subscriber

```
#include <Arduino.h>
#include <WiFi.h>

extern “C"
{
    #include "zenoh-pico.h"
}

void data_handler(const zn_sample_t *sample, const void *arg)
{
   // printf(">> [Subscription listener] Received (%.*s, %.*s)\n",
   //     (int)sample->key.len, sample->key.val,
   //     (int)sample->value.len, sample->value.val);
}

void setup()
{
    WiFi.mode(WIFI_STA);
    WiFi.begin(“MY_SSID", “MY_PASS");
    while (WiFi.status() != WL_CONNECTED)
        delay(1000);
```

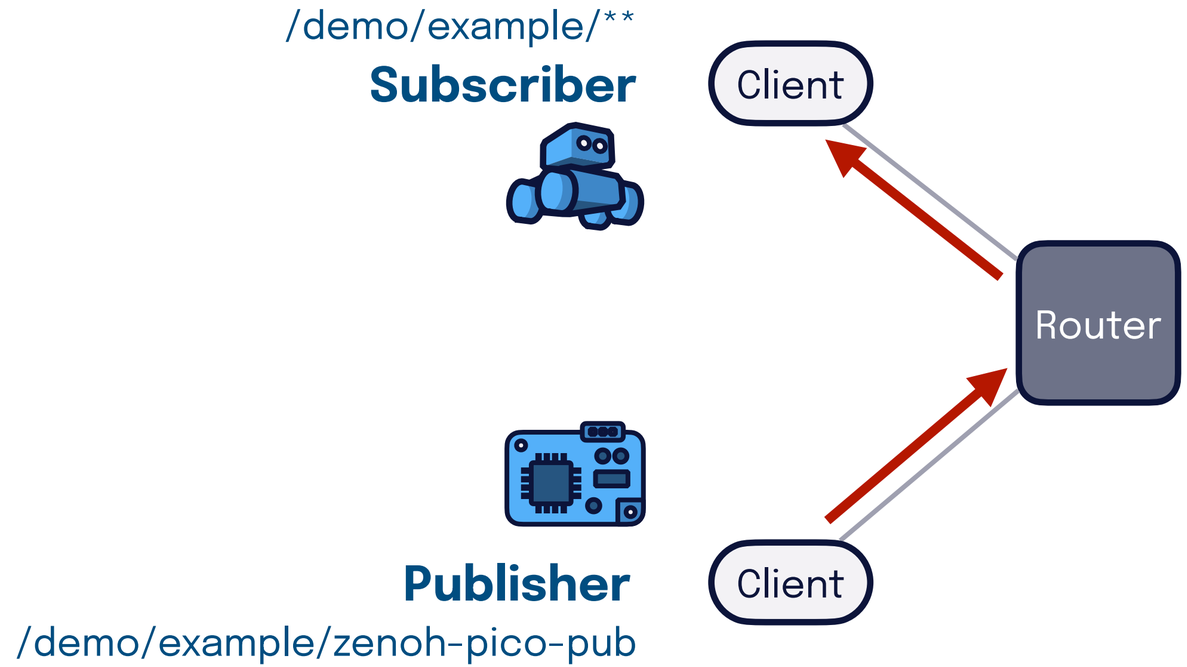
Continue… =>

```
    zn_properties_t *config = zn_config_default();
    zn_properties_insert(config, ZN_CONFIG_MODE_KEY,
        z_string_make(“peer”));
    zn_properties_insert(config, ZN_CONFIG_PEER_KEY,
        z_string_make(“udp/224.0.0.225:7447#iface=eth0”));

    zn_session_t *s = zn_open(config);
    if (s == NULL)
        return;

    znp_start_read_task(s);
    znp_start_lease_task(s);

    zn_reskey_t reskey = zn_rname("/demo/example/**");
    zn_subscriber_t *sub = zn_declare_subscriber(s, reskey,
        zn_subinfo_default(), data_handler, NULL);
    if (sub == 0)
        return;
}

void loop()
{
    delay(5000);
}
```

Note: Tutorial using Zenoh v0.5.0-beta9

Sub example: https://github.com/eclipse-zenoh/zenoh-pico/blob/master/examples/net/esp32/zn_sub.ino

Brokered communication



Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Publisher

```
#include <Arduino.h>
#include <WiFi.h>

extern "C"
{
    #include "zenoh-pico.h"
}

zn_session_t *s = NULL;
zn_reskey_t *reskey = NULL;

void setup()
{
    WiFi.mode(WIFI_STA);
    WiFi.begin(“MY_SSID", “MY_PASS");
    while (WiFi.status() != WL_CONNECTED)
        delay(1000);
```

Continue… =>

```
zn_properties_t *config = zn_config_default();
s = zn_open(config);
if (s == NULL)
    return;

znp_start_read_task(s);
znp_start_lease_task(s);

    unsigned long rid = zn_declare_resource(s,
        zn_rname("/demo/example/zenoh-pico-pub”));
    reskey = (zn_reskey_t *)malloc(sizeof(zn_reskey_t));
    *reskey = zn_rid(rid);
}

void loop()
{
    delay(5000);

    char *buf = "Publishing data from ESP32";
    zn_write(s, *reskey, (const uint8_t *)buf, strlen(buf));
}
```

Note: Tutorial using Zenoh v0.5.0-beta9

Pub example: https://github.com/eclipse-zenoh/zenoh-pico/blob/master/examples/net/esp32/zn_pub.ino

Subscriber

```
#include <Arduino.h>
#include <WiFi.h>

extern "C"
{
    #include "zenoh-pico.h"
}

void data_handler(const zn_sample_t *sample, const void *arg)
{
   // printf(">> [Subscription listener] Received (%.*s, %.*s)\n",
   //     (int)sample->key.len, sample->key.val,
   //     (int)sample->value.len, sample->value.val);
}

void setup()
{
    WiFi.mode(WIFI_STA);
    WiFi.begin(“MY_SSID", “MY_PASS");
    while (WiFi.status() != WL_CONNECTED)
        delay(1000);
```

Continue… =>

C

```
    zn_properties_t *config = zn_config_default();
    zn_session_t *s = zn_open(config);
    if (s == NULL)
        return;

    znp_start_read_task(s);
    znp_start_lease_task(s);

    zn_reskey_t reskey = zn_rname("/demo/example/**");
    zn_subscriber_t *sub = zn_declare_subscriber(s,
        reskey, zn_subinfo_default(), data_handler, NULL);
    if (sub == 0)
        return;
}

void loop()
{
    delay(5000);
}
```

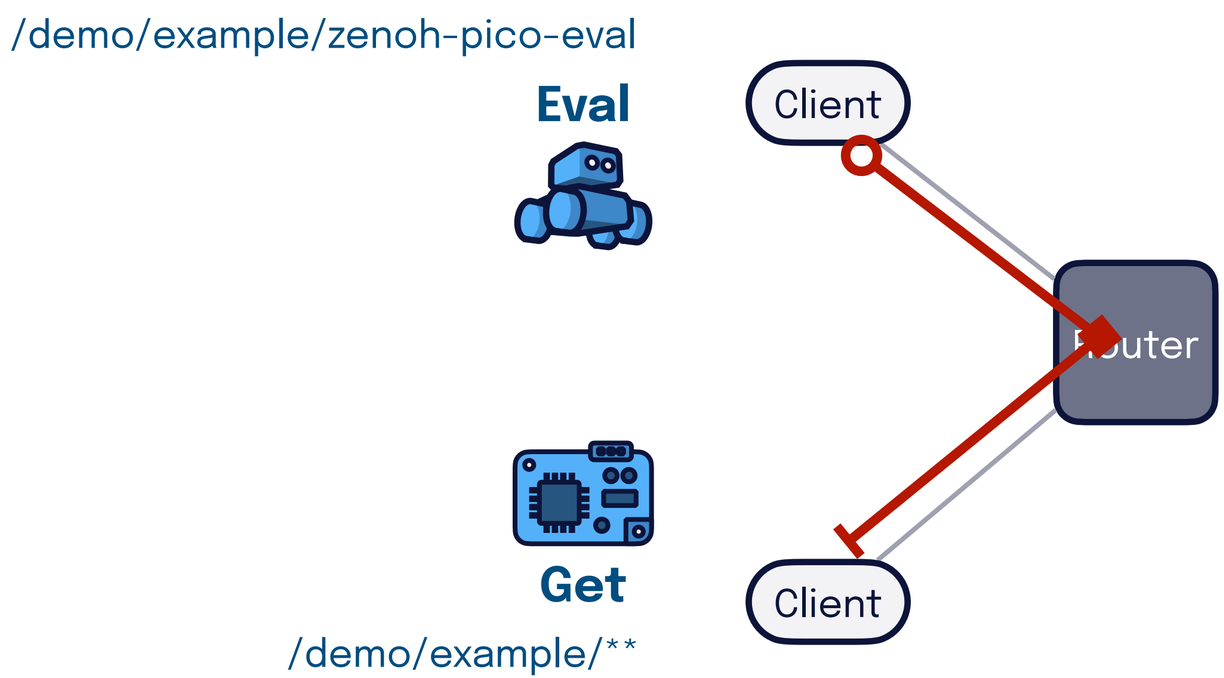
Note: Tutorial using Zenoh v0.5.0-beta9

Sub example: https://github.com/eclipse-zenoh/zenoh-pico/blob/master/examples/net/esp32/zn_sub.ino

zetta scale

Distributed computed values

Brokered communication



Note: Tutorial using Zenoh v0.5.0-beta9

```
Make sure zenohd is up and running
```

Get

```
#include <Arduino.h>
#include <WiFi.h>

extern "C"
{
    #include "zenoh-pico.h"
}

zn_session_t *s = NULL;

void setup()
{
    WiFi.mode(WIFI_STA);
    WiFi.begin(“MY_SSID", “MY_PASS");
    while (WiFi.status() != WL_CONNECTED)
        delay(1000);

    zn_properties_t *config = zn_config_default();
    zn_session_t *s = zn_open(config);
    if (s == NULL)
        return;

    znp_start_read_task(s);
    znp_start_lease_task(s);
}
```

Continue… =>

```
void loop()
{
    delay(5000);

    zn_reply_data_array_t replies = zn_query_collect(s,
        zn_rname("/demo/example/**"), "",
        zn_query_target_default(),
        zn_query_consolidation_default());

    for (unsigned int i = 0; i < replies.len; i++)
    {
        // printf(">> [Reply handler] received (%.*s, %.*s)\n",
        //        (int)replies.val[I].data.key.len,
        //        replies.val[I].data.key.val,
        //        (int)replies.val[i].data.value.len,
        //        replies.val[I].data.value.val);
    }
    zn_reply_data_array_free(replies);
}
```

Note: Tutorial using Zenoh v0.5.0-beta9

Get example: https://github.com/eclipse-zenoh/zenoh-pico/blob/master/examples/net/esp32/zn_get.ino

Eval

```
#include <Arduino.h>
#include <WiFi.h>

extern "C"
{
    #include "zenoh-pico.h"
}

void query_handler(zn_query_t *query, const void *arg)
{
    z_string_t res = zn_query_res_name(query);
    z_string_t pred = zn_query_predicate(query);
    zn_send_reply(query, query->rname,
        (const unsigned char *)but, strlen(but));
}

void setup()
{
    WiFi.mode(WIFI_STA);
    WiFi.begin("MY_SSID", "MY_PASS");
    while (WiFi.status() != WL_CONNECTED)
        delay(1000);
```

Continue… =>

C

```
    zn_properties_t *config = zn_config_default();
    zn_session_t *s = zn_open(config);
    if (s == NULL)
        return;

    znp_start_read_task(s);
    znp_start_lease_task(s);

    zn_reskey_t reskey = zn_rname(URI);
    zn_queryable_t *qable = zn_declare_queryable(s, reskey,
        ZN_QUERYABLE_EVAL, query_handler, NULL);
    if (qable == 0)
        return;
}

void loop()
{
    delay(5000);
}
```

Note: Tutorial using Zenoh v0.5.0-beta9

Eval example: https://github.com/eclipse-zenoh/zenoh-pico/blob/master/examples/net/esp32/zn_eval.ino

zetta scale

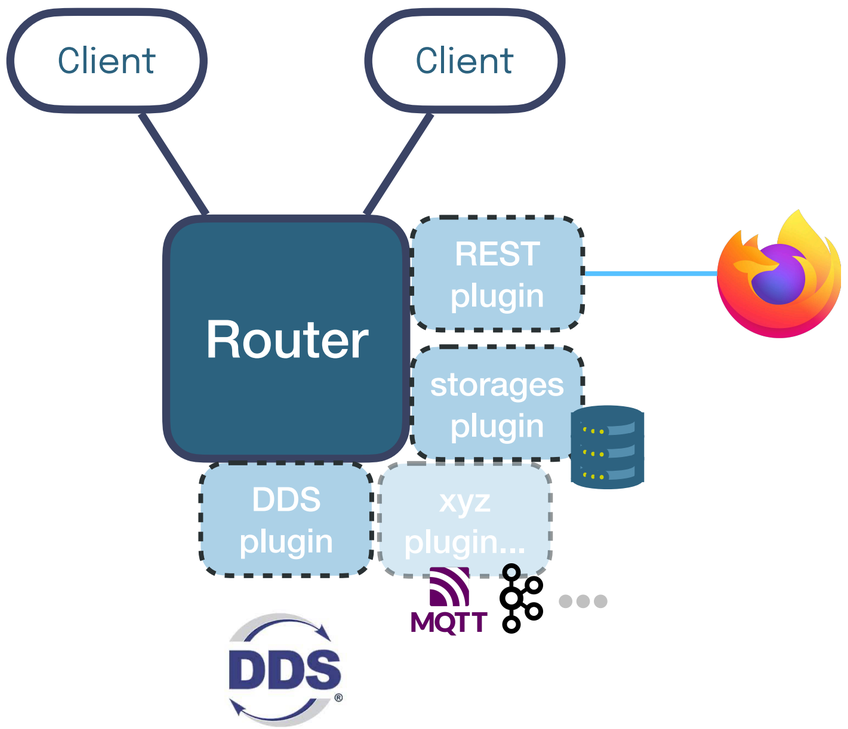
Zenoh plugin for DDS

The Zenoh team

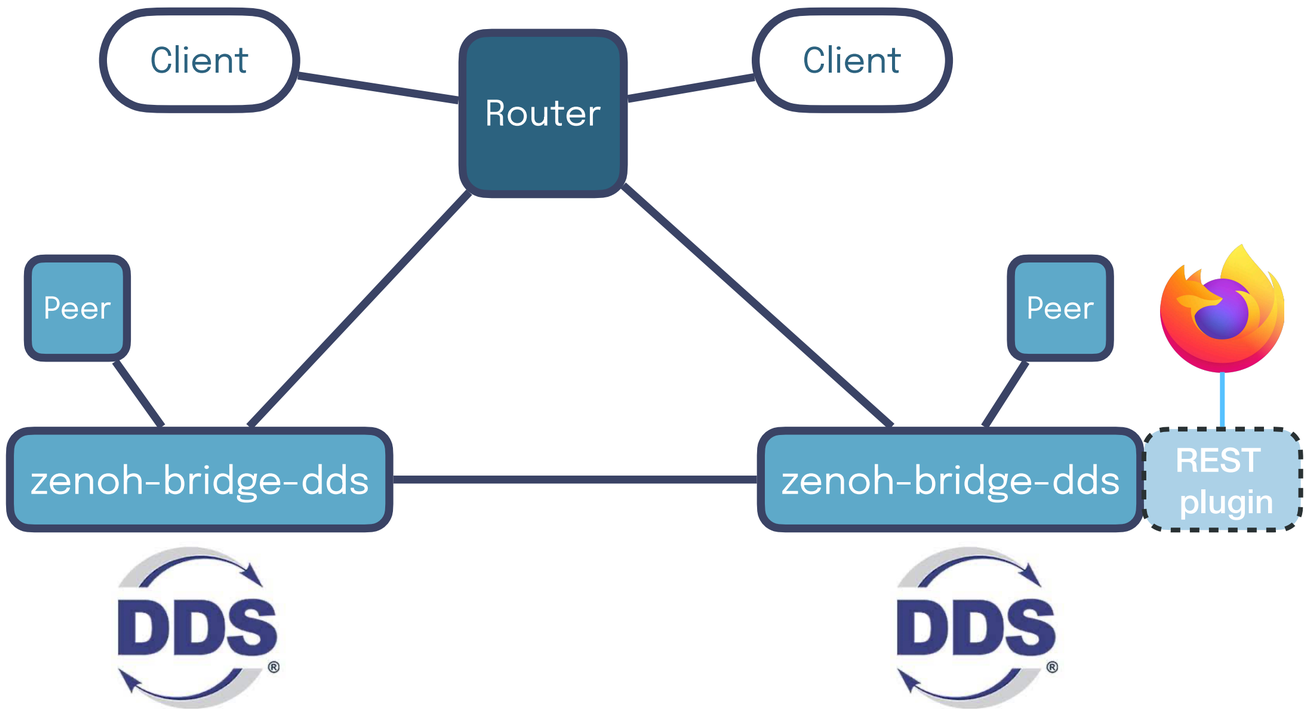
March 2nd, 2022

2 bundles

Plugin for zenoh router



Standalone process
(client or peer mode)



Note: Tutorial using Zenoh v0.5.0-beta9

How does it work ?

Default mode

- DDS topic => zenoh key "/topic" (or "</scope></partition>/topic" if any)
- No routing of DDS discovery
- At entity discovery, create a local matching entity

Note: Tutorial using Zenoh v0.5.0-beta9

How does it work ?

Forward discovery mode

- DDS topic => zenoh key "/topic" (or "</scope></partition>/topic" if any)
- Forwarding entities discovery info over zenoh (+ ros_discovery_info)
- At entity discovery, create a remote mirroring entity

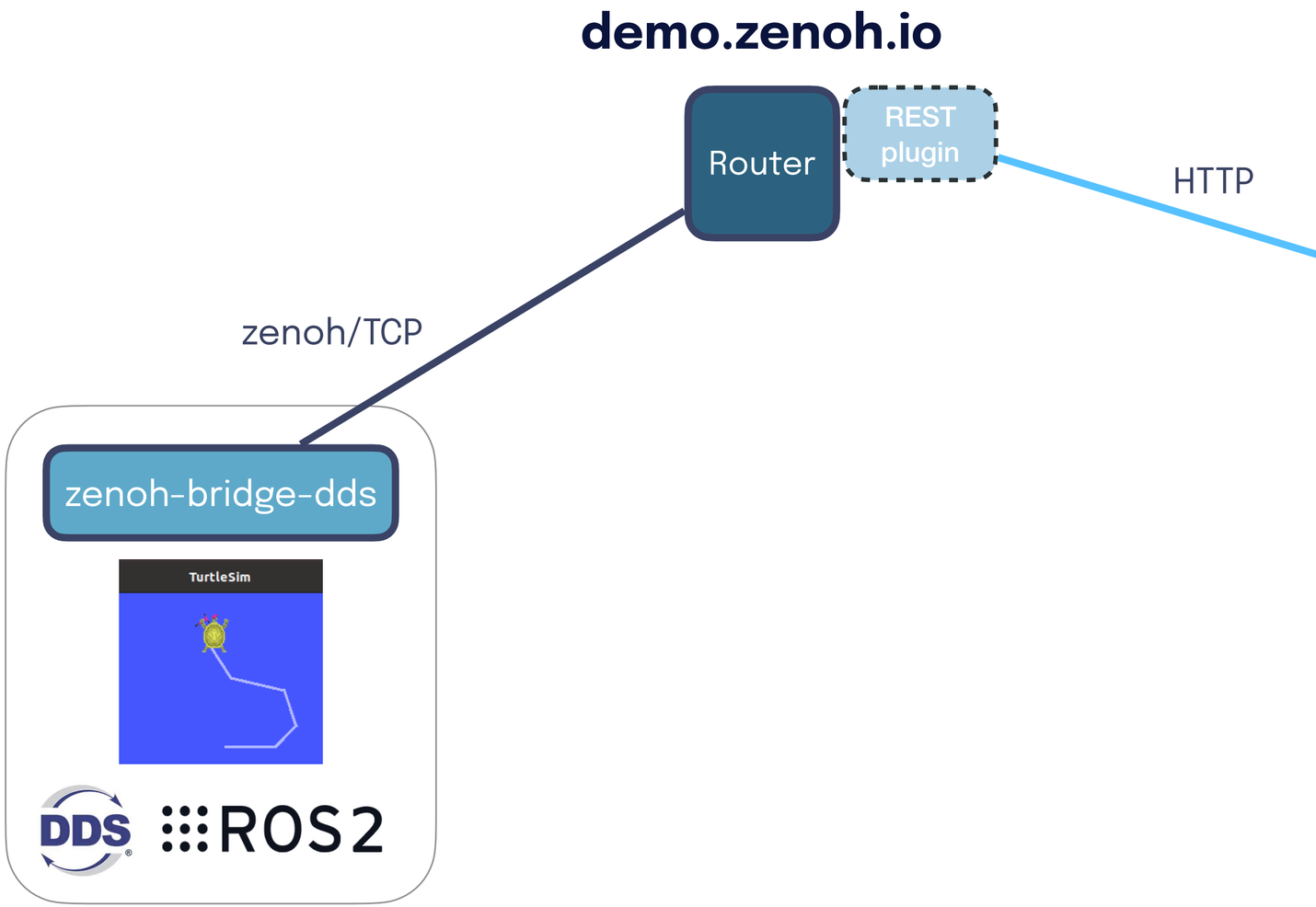
Note: Tutorial using Zenoh v0.5.0-beta9

Features

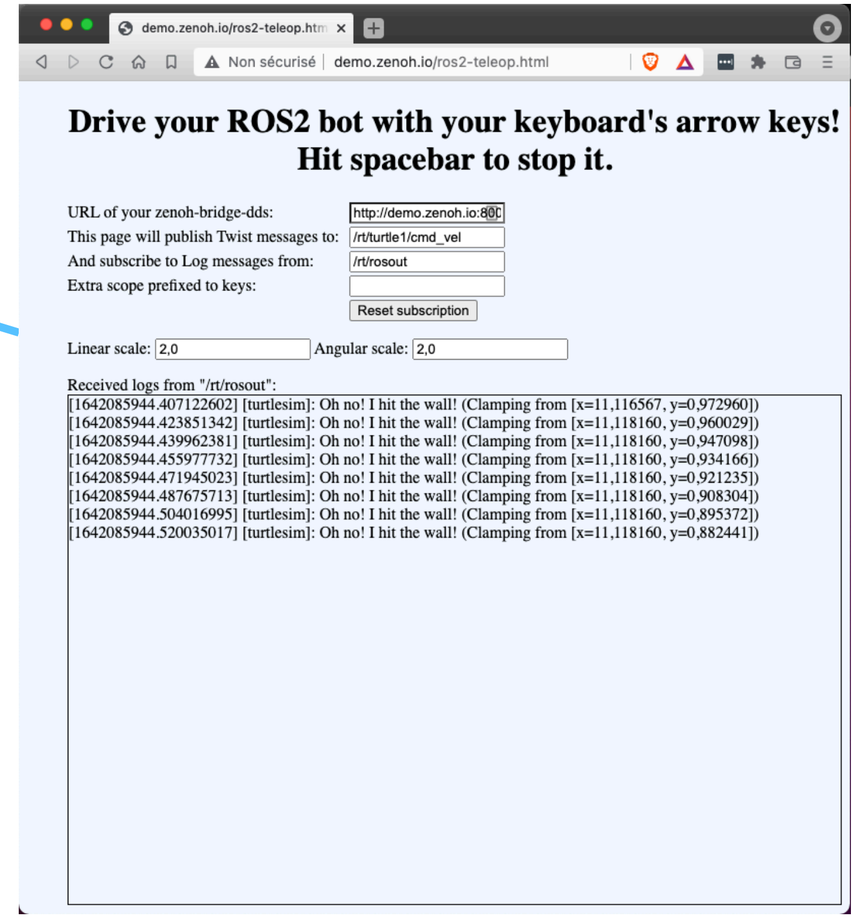
- Grouping of zenoh publisher/subscriber for minimal discovery traffic
- Configurable list of topics allowed to be routed
- Configurable per-topic downsampling to save bandwidth
- TRANSIENT_LOCAL support via caching, querying and group supervision
- Discovery and routing info via the zenoh Admin Space
- + All the zenoh features: TLS, UDP, QUIC, routing, fault tolerance...

Note: Tutorial using Zenoh v0.5.0-beta9

Demo



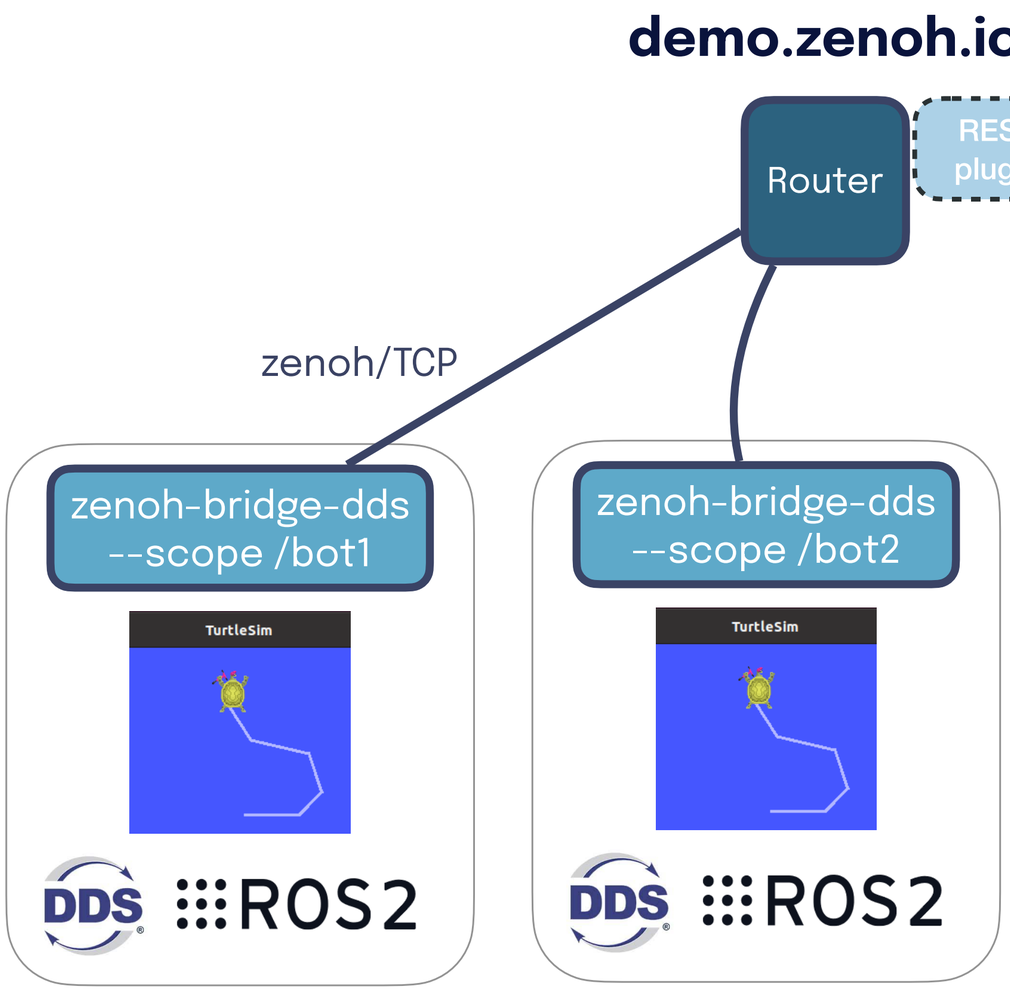
http://demo.zenoh.io/ros2-teleop.html

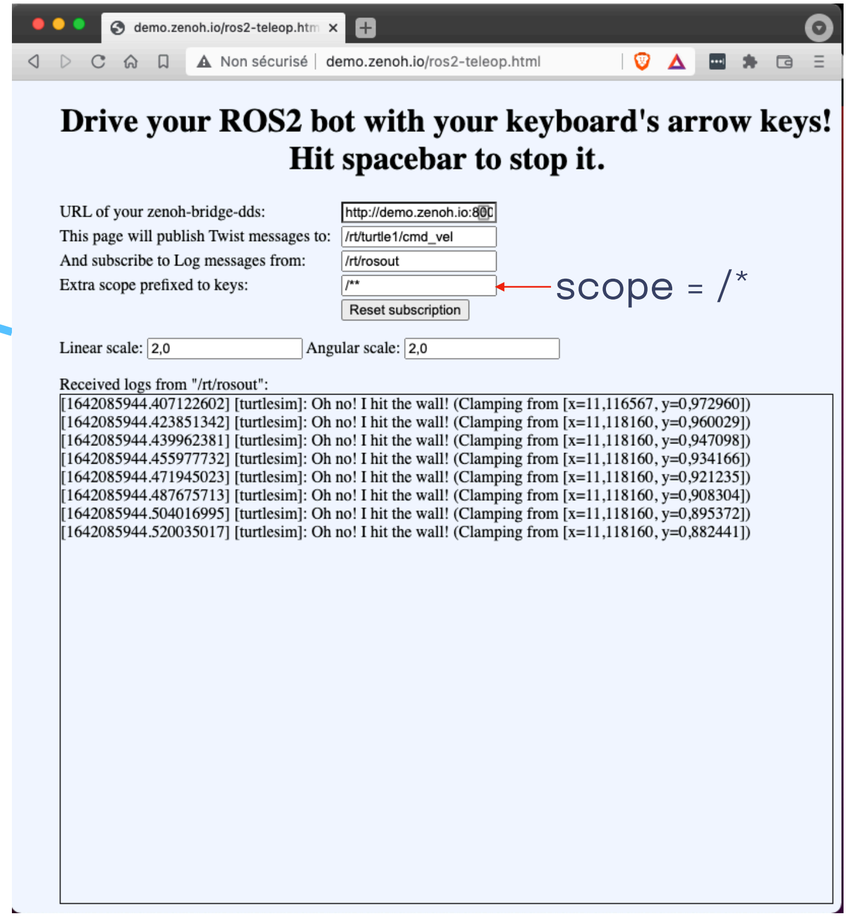


Note: Tutorial using Zenoh v0.5.0-beta9

Demo

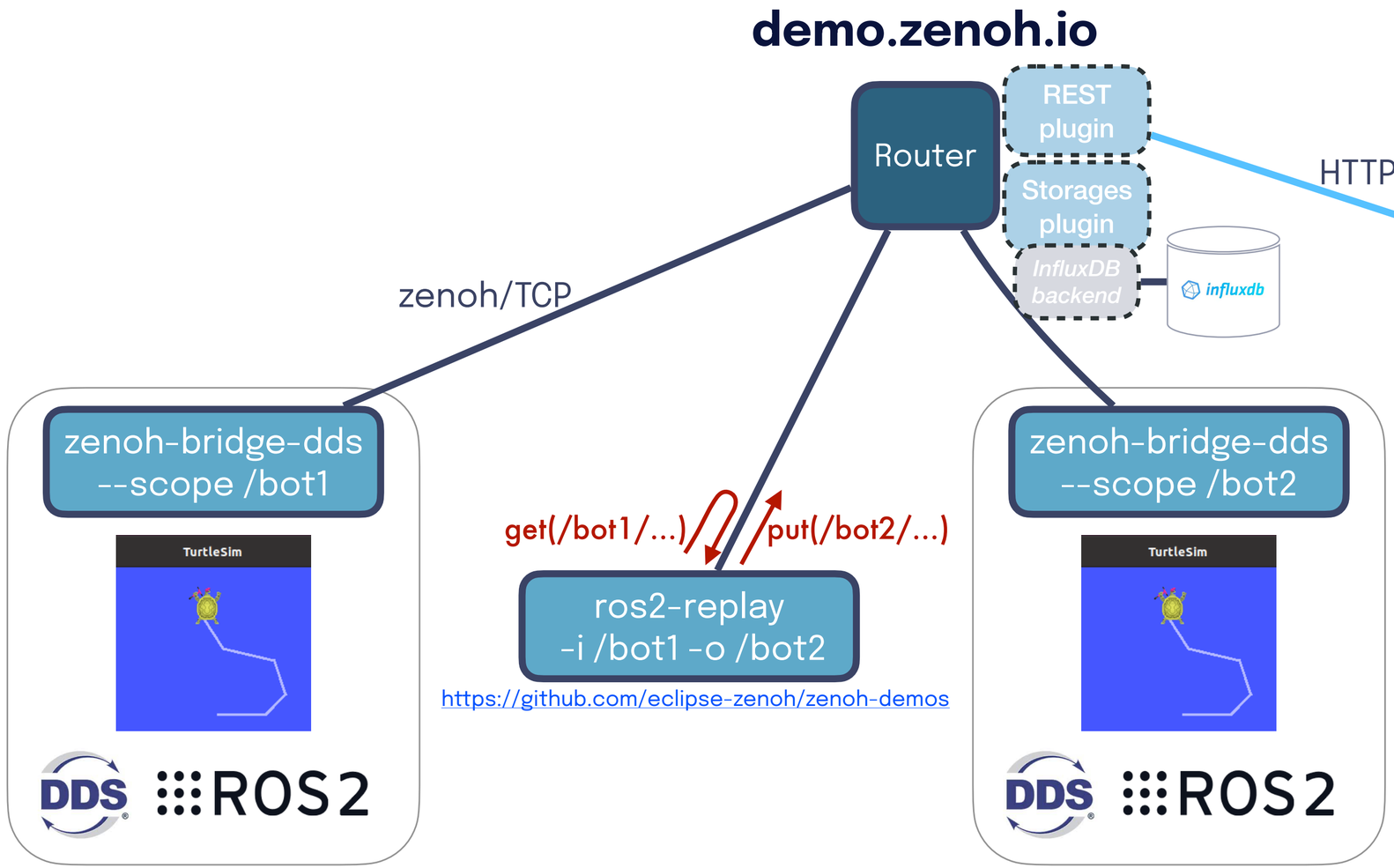
http://demo.zenoh.io/ros2-teleop.html



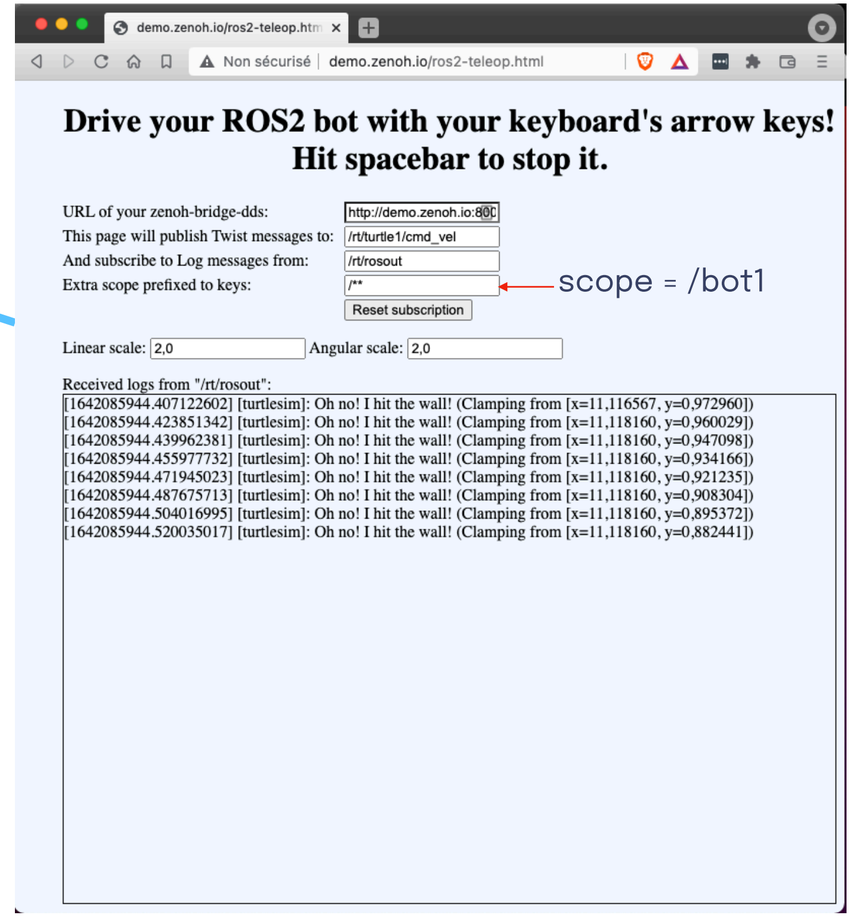


Note: Tutorial using Zenoh v0.5.0-beta9

Demo



http://demo.zenoh.io/ros2-teleop.html



Note: Tutorial using Zenoh v0.5.0-beta9

zetta scale

Zenoh Flow

Hands-on

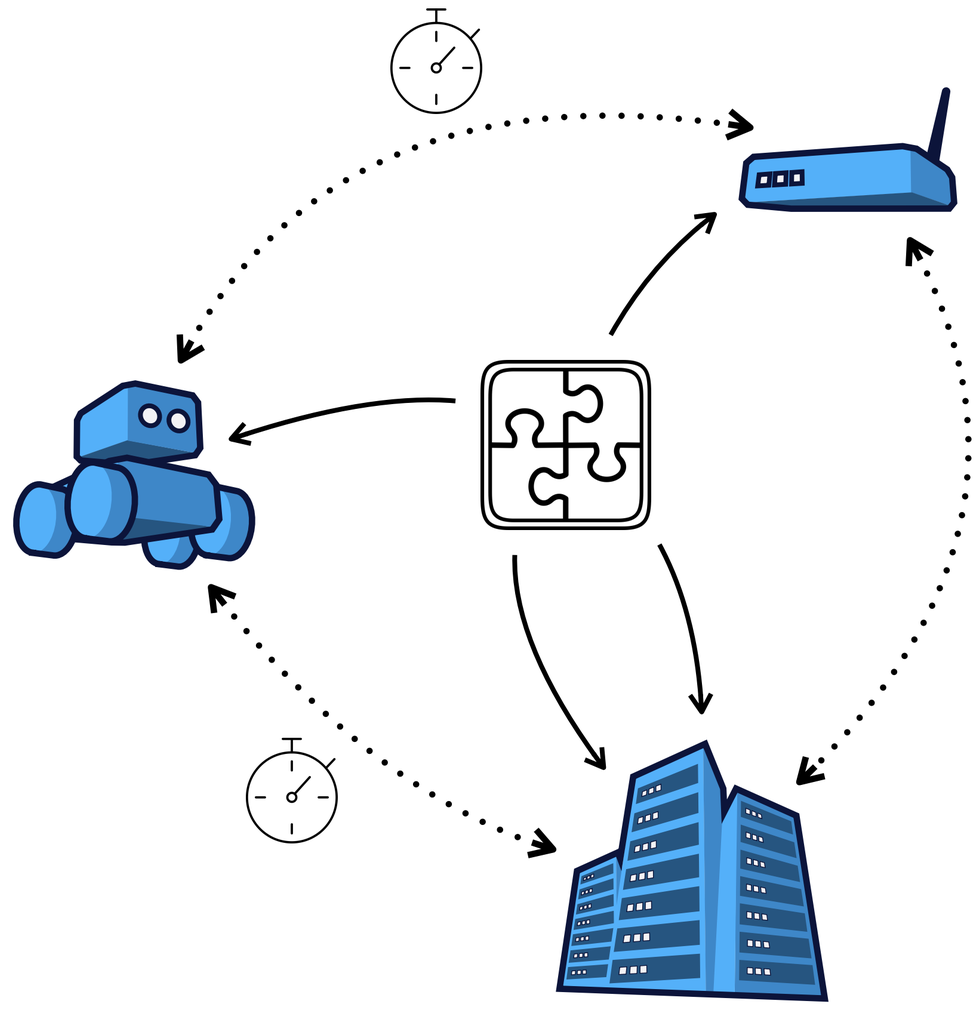
Julien Loudet, PhD — Product Conductor
julien.loudet@zettascale.tech
Gabriele Baldoni — Technologist
gabriele.baldoni@zettascale.tech

Zenoh Flow key concepts

Zenoh Flow is a **framework for the Cloud-to-Thing continuum**.

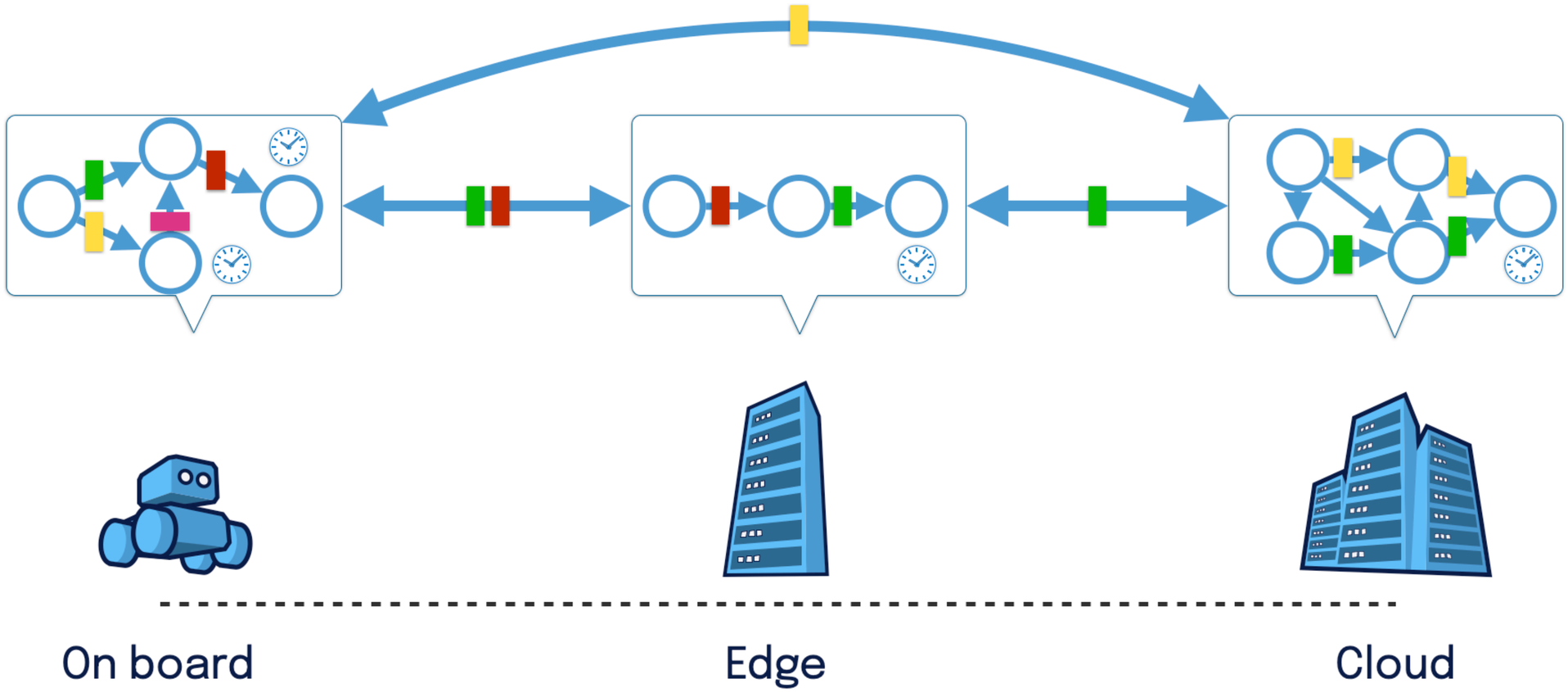
It leverages the **data flow programming model** to hide the **complexity** of distribution and decentralization.

It automatically manages the **underlying communications** and **deployment** of nodes.



Note: Tutorial using Zenoh-Flow v0.3.0

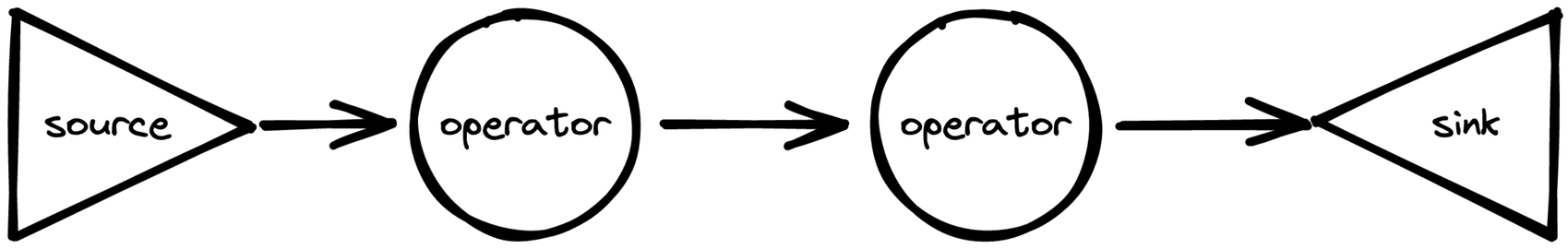
Zenoh Flow in a picture



zetta scale

Understanding Zenoh Flow

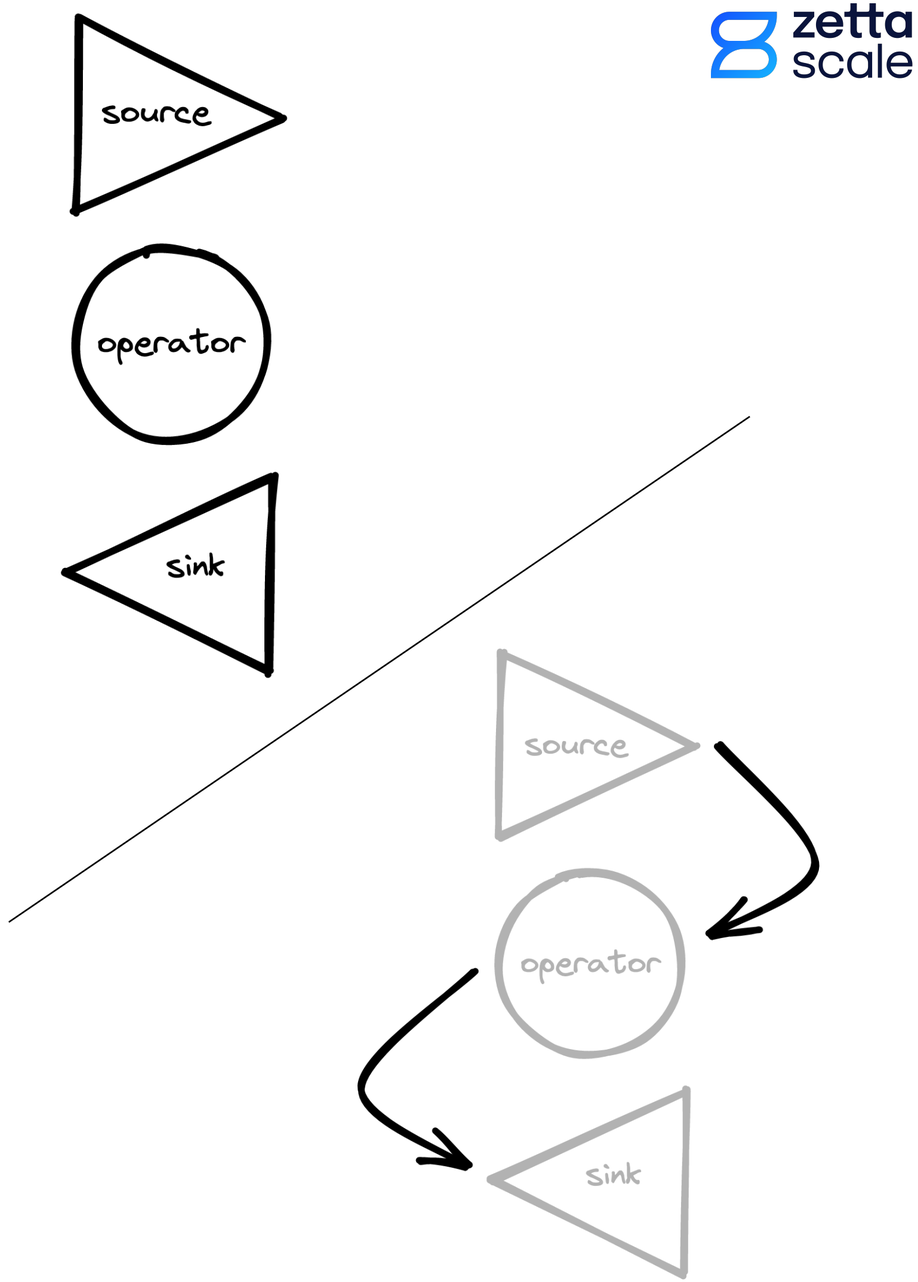
Data flow programming



Data flow programming represents **computations as a directed graph** of “nodes” that can execute **concurrently**.

It allows developers to **focus on the application logic**: the nodes and their connections.

Zenoh Flow: a three-steps process



Application developers need to:

1. Implement the “nodes” they need.
2. Describe the data flow graph.
3. Deployment.

Implementing nodes

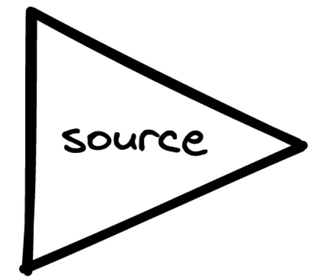
“The first step is to say that you can.” – Will Smith

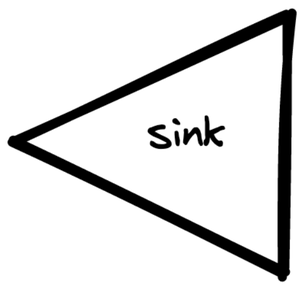
Nodes

There are 3 types of nodes:

- **source** : *fetching data,*
- **operator** : *computing over data,*
- **sink** : *sending out data.*

Nodes can be developed in Rust, C++, Python.





Nodes: semi-stateful

- Zenoh Flow offers the possibility for nodes to store a **state**.
- However, they must manage it themselves.

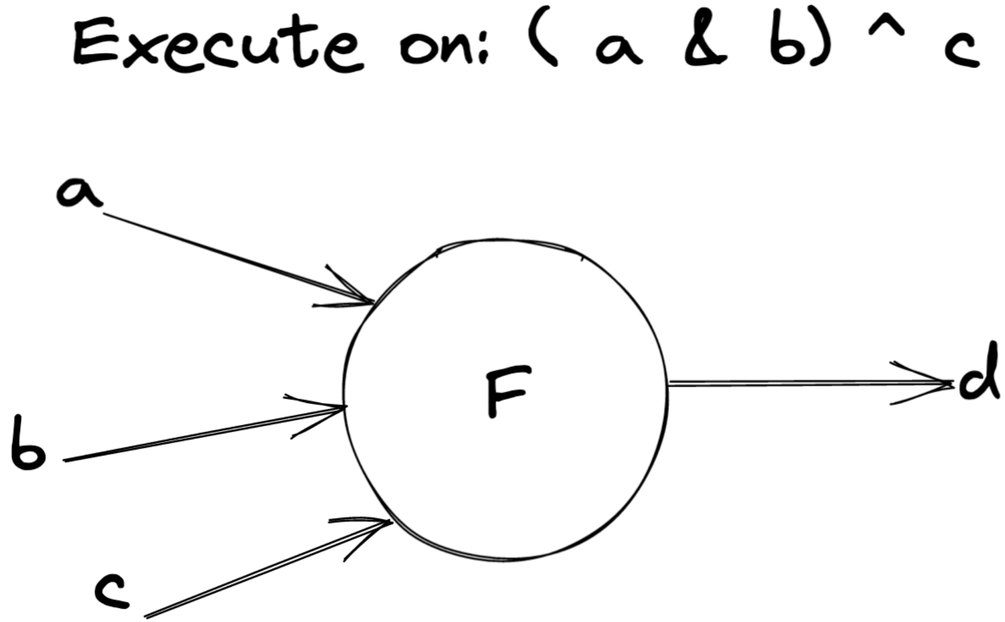
⚠️ *If a node fails, its state is lost.*

Input Rules

Input Rules **decide** when an *operator* **executes**.

It allows **synchronizing** inputs.

Application developers decide whether data should be **used**, **kept** for next executions or **dropped**.

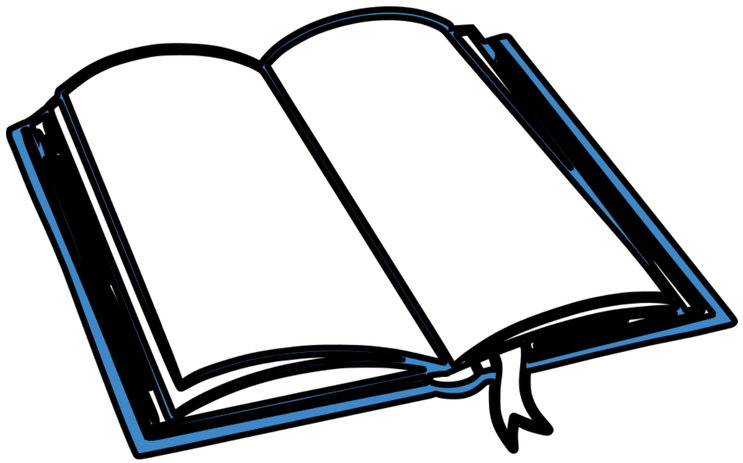


Dynamic “loading"

The “result” of the implementation is a:

- *shared library* (Rust / C++)
- *script* (Python)

that will later be **dynamically loaded** or **interpreted**.



Describing the data flow

“He who cannot describe the problem,
will never find the solution to that problem.” – Confucius

Data flow graph

The data flow graph is described in a **YAML** file. This file acts as a **contract** that is enforced by Zenoh Flow.

```yaml
flow: Robot
sources:
  - id: Camera
  - id: LIDAR
  - id: Network

operators:
  - id: Logic

sinks:
  - id: Motor
```

```yaml
links:
  - from:
      node: Camera
    to:
      node: Logic
  - from:
      node: LIDAR
    to:
      node: Logic
  - from:
      node: Network
    to:
      node: Logic
  - from:
      node: Logic
    to:
      node: Motor
```

Note: Tutorial using Zenoh-Flow v0.3.0

Data flow graph: operators

- **uri**: the path where the shared library / script can be found.
- **inputs**: the data the operator receives.
- **outputs**: the data the operator produces.

```
flow: Robot
sources:
  - id: Camera
  - id: LIDAR
  - id: Network

operators:
  - id: Logic
    uri: file://librobot.so
    inputs:
      - id: camera-stream
        type: frame
      - id: lidar-surroundings
        type: pointcloud
      - id: network-data
        type: bytes
    outputs:
      - id: command
        type: velocity

sinks:
  - id: Motor
```

zettascale

```
links:
  - from:
      node: Camera
    to:
      node: Logic
  - from:
      node: LIDAR
    to:
      node: Logic
  - from:
      node: Network
    to:
      node: Logic
  - from:
      node: Logic
    to:
      node: Motor
```

Note: Tutorial using Zenoh-Flow v0.3.0

Data flow graph: sources

Sources, as part of their interface, only have one **output**.

```
flow: Robot
sources:
  - id: Camera
    uri: file://libcamera.so
    output:
      id: stream
      type: frame
  - id: LIDAR
    uri: file://liblidar.so
    output:
      id: surroundings
      type: pointcloud
  - id: Network
    uri: file://libnetwork.so
    output:
      id: data
      type: bytes

operators:
  - id: Logic
    uri: file://librobot.so
    inputs:
      - id: camera-stream
        type: frame
      - id: lidar-surroundings
        type: pointcloud
      - id: network-data
        type: bytes
    outputs:
      - id: command
        type: velocity

sinks:
  - id: Motor
```

```
links:
  - from:
      node: Camera
    to:
      node: Logic
  - from:
      node: LIDAR
    to:
      node: Logic
  - from:
      node: Network
    to:
      node: Logic
  - from:
      node: Logic
    to:
      node: Motor
```

Note: Tutorial using Zenoh-Flow v0.3.0

Data flow graph: sinks

Similarly, *Sinks*, as part of their interface, only have one **input**.

```
flow: Robot
sources:
  - id: Camera
    uri: file://libcamera.so
    output:
      id: stream
      type: frame
  - id: LIDAR
    uri: file://liblidar.so
    output:
      id: surroundings
      type: pointcloud
  - id: Network
    uri: file://libnetwork.so
    output:
      id: data
      type: bytes

operators:
  - id: Logic
    uri: file://librobot.so
    inputs:
      - id: camera-stream
        type: frame
      - id: lidar-surroundings
        type: pointcloud
      - id: network-data
        type: bytes
    outputs:
      - id: command
        type: velocity

sinks:
  - id: Motor
    uri: file://libmotor.so
    input:
      id: logic-command
      type: velocity
```

zettascale

```
links:
  - from:
      node: Camera
    to:
      node: Logic
  - from:
      node: LIDAR
    to:
      node: Logic
  - from:
      node: Network
    to:
      node: Logic
  - from:
      node: Logic
    to:
      node: Motor
```

Note: Tutorial using Zenoh-Flow v0.3.0

Data flow graph: links

Links specify how nodes are connected.

⚠️ All inputs and outputs must be connected.

```yaml
flow: Robot
sources:
  - id: Camera
    uri: file://libcamera.so
    output:
      id: stream
      type: frame
  - id: LIDAR
    uri: file://liblidar.so
    output:
      id: surroundings
      type: pointcloud
  - id: Network
    uri: file://libnetwork.so
    output:
      id: data
      type: bytes

operators:
  - id: Logic
    uri: file://librobot.so
    inputs:
      - id: camera-stream
        type: frame
      - id: lidar-surroundings
        type: pointcloud
      - id: network-data
        type: bytes
    outputs:
      - id: command
        type: velocity

sinks:
  - id: Motor
    uri: file://libmotor.so
    input:
      id: logic-command
      type: velocity
```

zetta scale

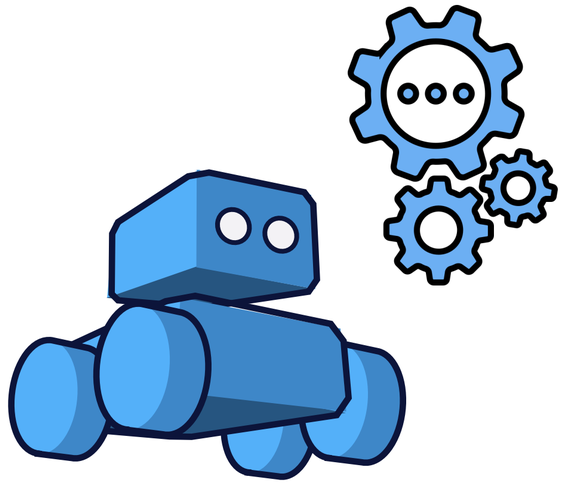
```yaml
links:
  - from:
      node: Camera
      output: stream
    to:
      node: Logic
      input: camera-stream
  - from:
      node: LIDAR
      output: surroundings
    to:
      node: Logic
      input: lidar-surroundings
  - from:
      node: Network
      output: data
    to:
      node: Logic
      input: network-data
  - from:
      node: Logic
      output: command
    to:
      node: Motor
      input: logic-command
```

Deploying the data flow

“Until you spread your wings, you will have no idea how far you can fly.” – Napoleon

Deployment

Zenoh Flow **daemons** running on the target hardware deploy the “correct” nodes.



```
flow: Robot
sources:
  - id: Camera
    uri: file://libcamera.so
    output:
      id: stream
      type: frame
  - id: LIDAR
    uri: file://liblidar.so
    output:
      id: surroundings
      type: pointcloud
  - id: Network
    uri: file://libnetwork.so
    output:
      id: data
      type: bytes

operators:
  - id: Logic
    uri: file://librobot.so
    inputs:
      - id: camera-stream
        type: frame
      - id: lidar-surroundings
        type: pointcloud
      - id: network-data
        type: bytes
    outputs:
      - id: command
        type: velocity

sinks:
  - id: Motor
    uri: file://libmotor.so
    input:
      id: logic-command
      type: velocity
```

```
links:
- from:
    node: Camera
    output: stream
  to:
    node: Logic
    input: camera-stream
- from:
    node: LIDAR
    output: surroundings
  to:
    node: Logic
    input: lidar-surroundings
- from:
    node: Network
    output: data
  to:
    node: Logic
    input: network-data
- from:
    node: Logic
    output: command
  to:
    node: Motor
    input: logic-command
```

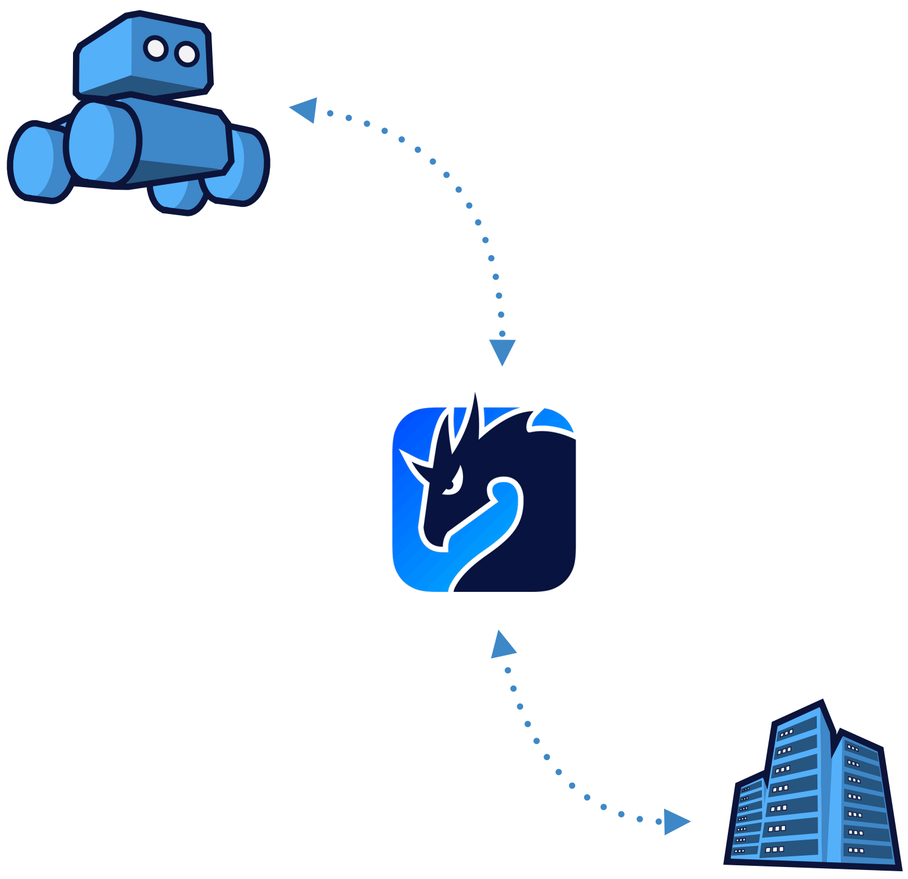
Deployment: mappings

In the YAML file, we (manually) specify which nodes run on which daemon.

🚧 *Automatic & "intelligent" deployment is planned next.*

```yaml
mappings:
  - id: camera
    daemon: robot
  - id: lidar
    daemon: robot
  - id: network
    daemon: robot
  - id: logic
    daemon: cloud
  - id: motor
    daemon: robot
```

Deployment: propagating the YAML file



Through Zenoh, Zenoh Flow daemons are aware of each other.

To deploy a data flow, we thus only need to forward the YAML file to one.

Try Zenoh Flow

Install Zenoh Flow

Install the dependencies and Zenoh

```
$ curl --proto "=https" --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- --default-toolchain stable -y
$ source $HOME/.cargo/env
$ git clone https://github.com/eclipse-zenoh/zenoh
$ cd zenoh && cargo build --release --all-targets && cd ..
```

Configure Zenoh

zenoh.json

```
{
    "listen": {
        "endpoints": ["tcp/0.0.0.0:7887"],
    },
    "plugins_search_dirs": ["/usr/lib"],
    "plugins": {
        "storages": {
            "required": true,
            "backends": {
                "memory": {
                    "required": true,
                    "storages": {
                        "zfrpc": {
                            "key_expr": "/zf/runtime/**"
                        },
                        "zf": {
                            "key_expr": "/zenoh-flow/**"
                        }
                    }
                }
            }
        }
    }
}
```

This configuration sets up the **Control Plane** that Zenoh Flow uses to *discover runtimes* and *deploy flows*.

Note: Tutorial using Zenoh-Flow v0.3.0

Install Zenoh Flow

Build Zenoh Flow daemon and zfctl

```
$ git clone https://github.com/eclipse-zenoh/zenoh-flow
$ cd zenoh-flow
$ cargo build --release --all-targets
$ sed -I ’s?\/etc/zenoh-flow\/extensions.d?'`pwd`\/zenoh-flow-daemon\/etc\/extension.d'?'
zenoh-flow-daemon/etc/runtime.yaml
```

Install Zenoh Flow

Build and install Zenoh Flow Python API 

```
$ git clone https://github.com/ZettaScaleLabs/zenoh-flow-python
$ cd zenoh-flow-python/zenoh-flow-python
$ pip3 install -r requirements-dev.txt
$ python3 setup.py bdist_wheel
$ pip3 install dist/*.whl
$ cd ..
$ cargo build --release -p py-source -p py-op -p py-sink
$ sed -i 's?\.?'`pwd`'?' 01-python.zfext
$ cp 01-python.zfext ../zenoh-flow/zenoh-flow-daemon/etc/extension.d/
```

Launch Zenoh Flow

zenohd

```
$ cd zenoh
$ RUST_LOG=info ./target/release/zenohd
-c zenoh.json
```

zenoh-flow-daemon

```
$ cd zenoh-flow
$ RUST_LOG=zenoh_flow=debug ./target/release/zenoh-
flow-daemon -c zenoh-flow-daemon/etc/runtime.yaml
```

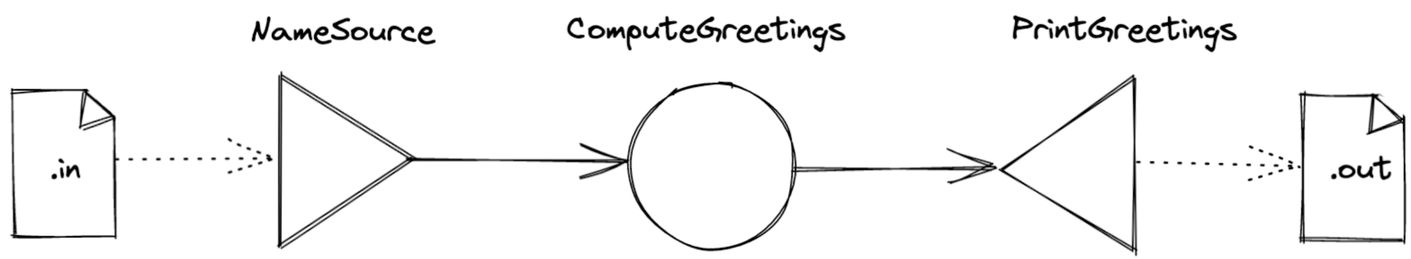
zfctl

```
$ cd zenoh-flow
$ ./target/release/zfctl list runtimes
```

Hello, Zenoh Flow!

This application is composed of three nodes:

- a *source*: **NameSource**,
- an *operator*: **ComputeGreetings**
- a *sink*: **PrintGreetings**.



The source **reads** the names from a file, and the sink **writes** the result to another file.

Hello, Zenoh Flow!

Prepare the workspace

```
$ cd
$ mkdir zf-helloworld
$ cd zf-helloworld
```

Hello, Zenoh Flow!

src-name.py

```
from zenoh_flow import Inputs, Outputs, Source
import time

class InputData:
    def __init__(self, filename="/tmp/helloworld-zf.in"):
        self.file = open(filename, "w+")
    def close(self):
        self.file.close()


class NameSource(Source):
    def initialize(self, configuration):
        return InputData()

    def finalize(self, state):
        return state.close()

    def run(self, _ctx, state):
        name = state.file.read()
        return str_to_bytes(name.rstrip("\n"))


# commodity function for serialization
def str_to_bytes(x: str) -> bytes:
    return x.encode('utf-8')

# Function that "exports" the source
def register():
    return NameSource
```

Reads data from a file.

Returns the data to Zenoh Flow, **injecting** it in the data flow.

Note: Tutorial using Zenoh-Flow v0.3.0

Hello, Zenoh Flow!

sink-greeting.py

```
from zenoh_flow import Sink

class OutputData:
    def __init__(self, filename="/tmp/helloworld-zf.out"):
        self.file = open(filename, "w+")
    def close(self):
        self.file.close()

class PrintGreetings(Sink):

    def initialize(self, configuration):
        return OutputData()

    def finalize(self, state):
        return state.close()

    def run(self, _ctx, state, input):
        greetings = f"{bytes_to_str(input.data)}\n"
        if len(greetings) == '':
            return None
        state.file.write(greetings)
        state.file.flush()

# commodity function for deserialization
def bytes_to_str(x: bytes) -> str:
    return x.decode('utf-8')

# Function that "exports" the sink
def register():
    return PrintGreetings
```

Receives data from a Zenoh Flow.

Writes the data into a file.

Note: Tutorial using Zenoh-Flow v0.3.0

Hello, Zenoh Flow!

op-greetings.py

```
from zenoh_flow import Inputs, Operator, Outputs


class ComputeGreetings(Operator):

    def initialize(self, configuration):
        return None

    def finalize(self, state):
        return None

    def input_rule(self, _ctx, state, tokens):
        return True

    def output_rule(self, _ctx, _state, outputs, _deadline_miss):
        return outputs

    def run(self, _ctx, _state, inputs):
        # Getting the inputs
        data = inputs.get('Name').data

        # Computing over the inputs
        name = bytes_to_str(data)
        if len(name) == 0:
            return {}

        greetings = f"Hello, {name}!"
        # Producing the outputs
        outputs = {'Greetings' : str_to_bytes(greetings)}
        return outputs

# Commodity for serialization/deserialization
def bytes_to_str(x: bytes) -> str:
    return x.decode('utf-8')
def str_to_bytes(x: str) -> bytes:
    return x.encode('utf-8')

# Function that "exports" the operator
def register():
    return ComputeGreetings
```

Receives data from a Zenoh Flow.

Performs the **computation**.

Returns the new data to Zenoh Flow, **sending** it to downstream nodes.

Note: Tutorial using Zenoh-Flow v0.3.0

Hello, Zenoh Flow!

hello-world.yml

```
flow: HelloWorld
operators:
  - id : GenGreeting
    uri: file:///home/<your user>/zf-helloworld/op-greetings.py
    inputs:
      - id: Name
        type: str
    outputs:
      - id: Greetings
        type: str
sources:
  - id : ReadName
    uri: file:///home/<your user>/zf-helloworld/src-name.py
    output:
      id: Name
      type: str
sinks:
  - id : PrintGreetings
    uri: file:///home/<your user>/zf-helloworld/sink-greetings.py
    input:
      id: Greetings
      type: str

links:
- from:
    node : ReadName
    output : Name
  to:
    node : GenGreeting
    input : Name
- from:
    node : GenGreeting
    output : Greetings
  to:
    node : PrintGreetings
    input : Greetings
```

Describes the application.

Lists sources, sink and operators.

Specifies the connection between them.

Note: Tutorial using Zenoh-Flow v0.3.0

Hello, Zenoh Flow!

zenohd

```
$ cd zenoh
$ RUST_LOG=info ./target/release/zenohd -c
zenoh.json
```

zenoh-flow-daemon

```
$ cd zenoh-flow
$ RUST_LOG=zenoh_flow=debug ./target/release/zenoh-
flow-daemon -c zenoh-flow-daemon/etc/runtime.yaml
```

zfctl

```
$ cd zenoh-flow
$ ./target/release/zfctl launch ~/zf-helloworld/hello-world.yml
```

Hello, Zenoh Flow!

Input

$ echo “Zenoh Flow” >> /tmp/helloworld-zf.in

Output

$ tail -f /tmp/helloworld-zf.out

Writing data into the input file will **trigger** the computation.

As **result** data will be written to the output file.

Note: Tutorial using Zenoh-Flow v0.3.0

Hello, Zenoh Flow!

Shutdown the application

```
$ cd zenoh-flow
$ ./target/release/zfctl destroy <instance uuid>
```

zetta scale

Thank You

Find out more in our channels