EPI Framework Demo

Tim Müller (CCI Group, UvA)

t.muller@uva.nl

UMC Utrecht - 23 October 2023

Schedule

- 14:30 - 15:05: **Hello, world! in Brane** *(guided hands-on)*
- 15:05 - 15:25: **EPIF in the PoC** *(presentation)*
- 15:25 - 15:30: **Questions, thoughts, evaluation, …**

Hello, world! in Brane (guided hands-on)

- Write your first **Hello, world!-package**!
- See the steps at https://wiki.enablingpersonalizedinterventions.nl/user-guide
  - Bottom-left, scroll down to “**35. Tutorials**”, then “**35.2.1. Hands-on session: Hello, world!**”
  - Or see: https://tinyurl.com/umc-utrecht-demo
- I’ll go through it on the board!

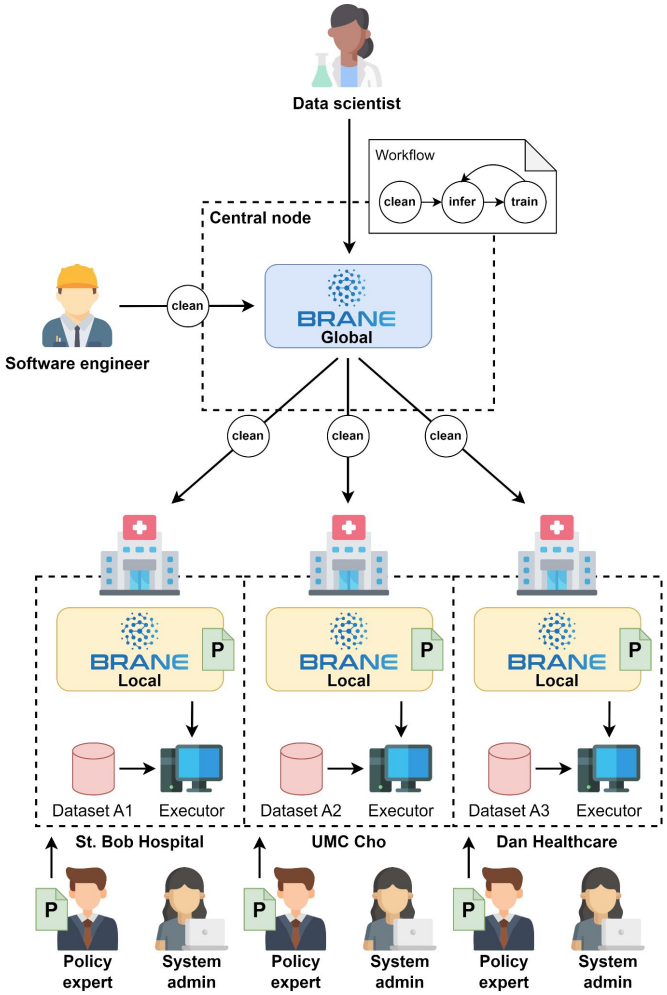
EPIF in the PoC

Or: The EPIF admin-side

I. Proof-of-Concept (PoC)

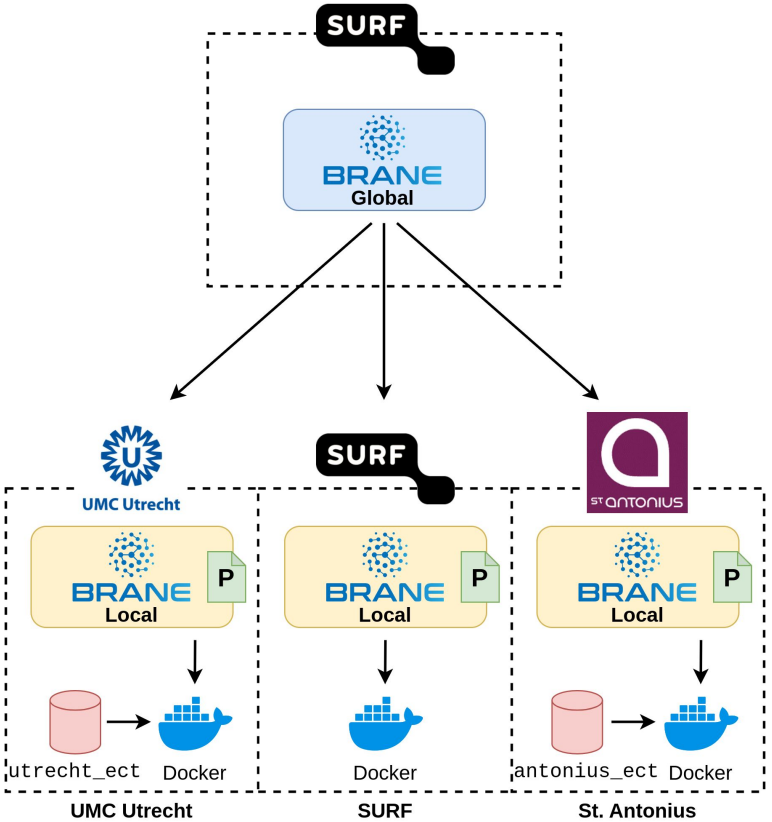
Where we left off…

- The **EPI Framework** is a:
  “Federated workflow execution engine”
- We’ve discussed using the framework
  - Data scientist
  - Software engineer
  - Policy expert
- Now: **PoC-specifics** as a **system admin**



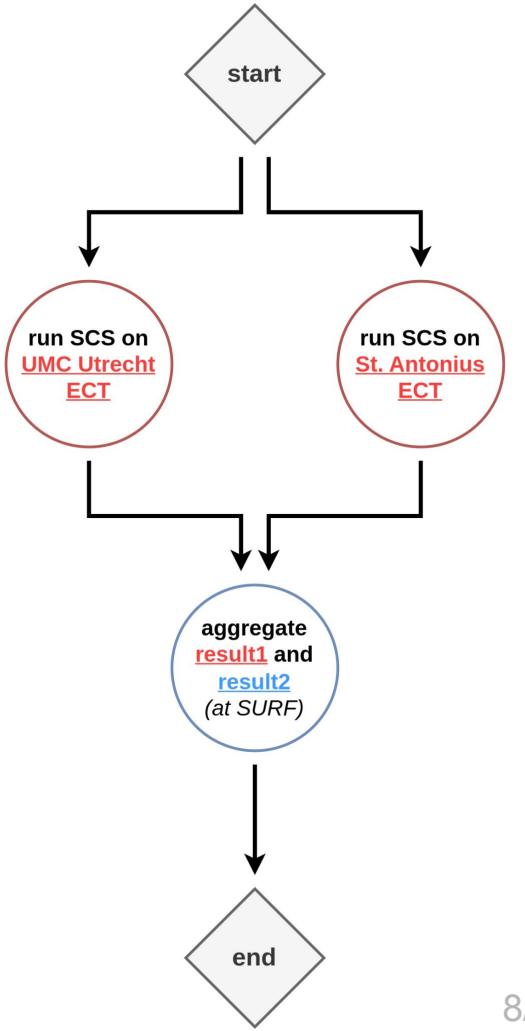
PoC - The EPIF-perspective

- One **central node**
  - Hosted by SURF
- Three **worker nodes**
  - SURF (aggregation)
  - St. Antonius, UMC Utrecht (local compute)
- Two **datasets**
  - `umc_utrecht_ect`
  - `st_antonius_ect`
- Two **use-cases**
  - **Rosanne's use-case** (stratified confidence)
  - **Saba's use-case** (synthetic data)



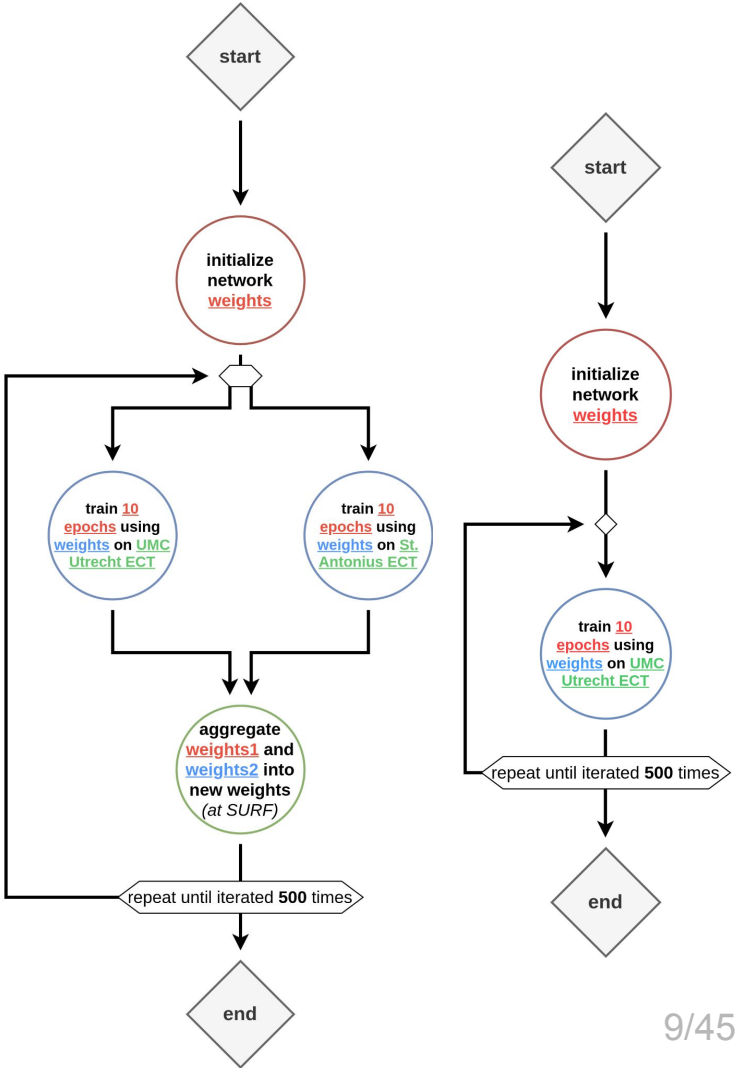
PoC - Rosanne's use-case

- **Stratified Confidence Sequence** (SCS) analysis
- **Federated** analysis
  - **Compute** SCS **locally** (UMC Utrecht, St. Antonius)
  - Send to **Trusted Third-Party** (TTP) (SURF)
  - Aggregate into **global result**
- Result: single value (number)

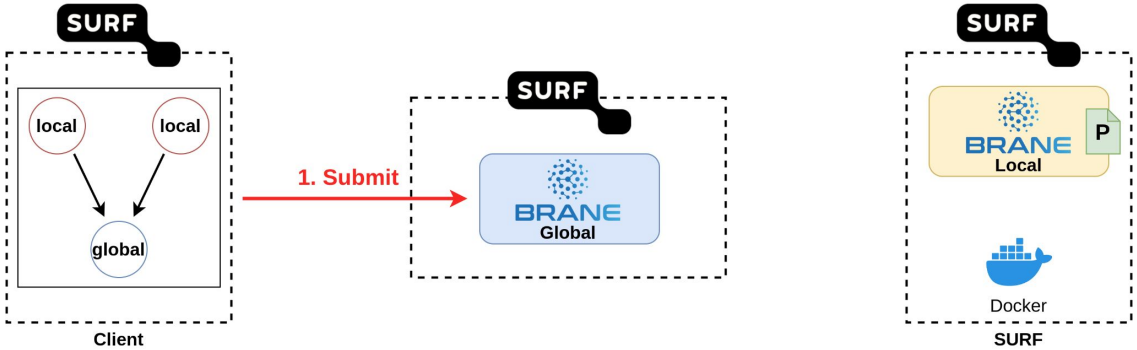


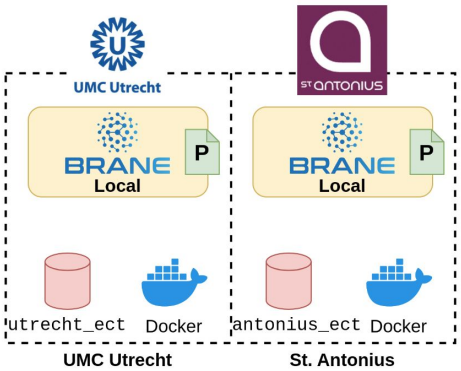
PoC - Saba's use-case

- Training a **synthetic generation algorithm**
  - First train as **federated algorithm**
  - Then generate new set from **central algorithm**
- Training simple validation **neural network**
  - Once as **federated algorithm** on raw data
  - Twice as **centralised algorithm** on half data
  - Once on **synthetic data** (also centralised)
- Result: various trained NN models (weights)
  - Different hyperparameters (number of iterations, hidden layers)
  - Compare using a test set (20% of data)

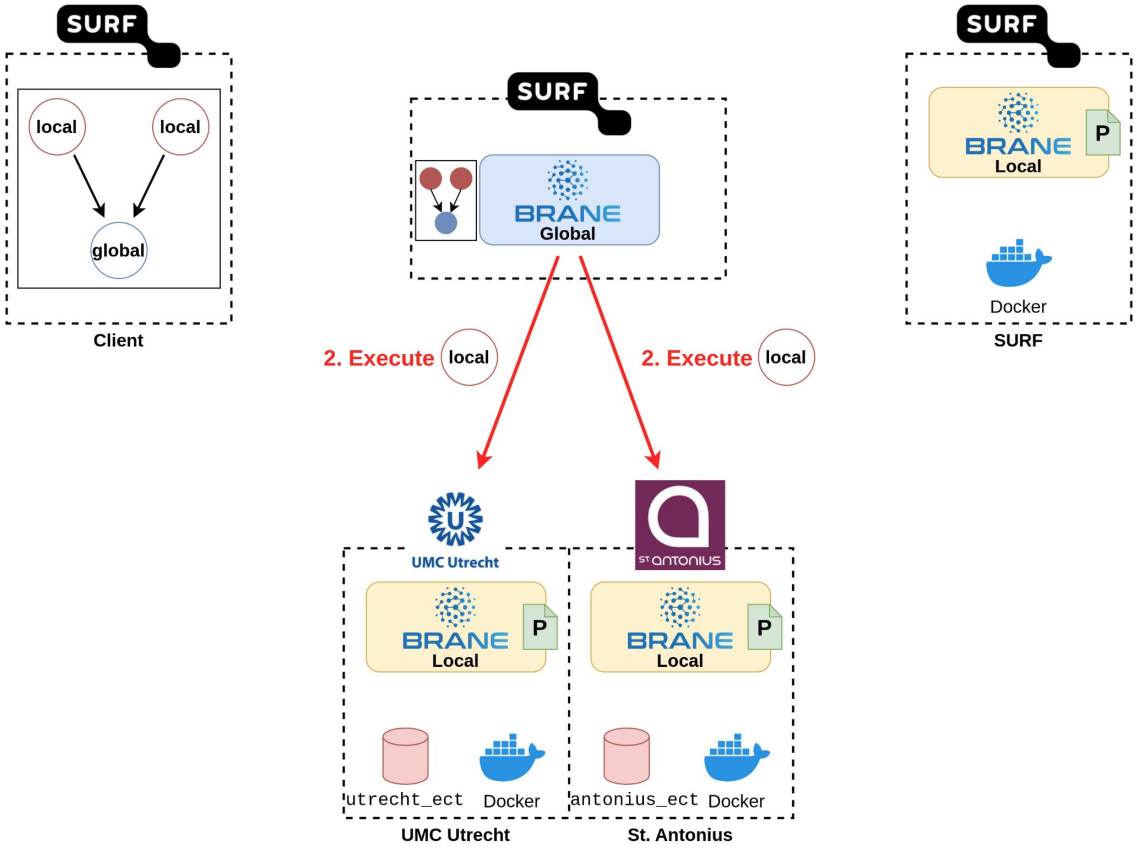


PoC in action - Rosanne's use-case

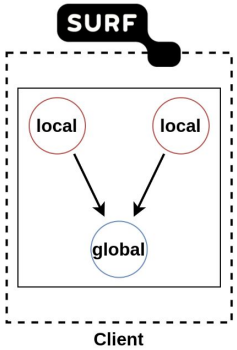


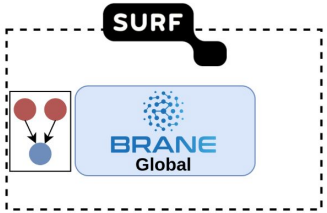


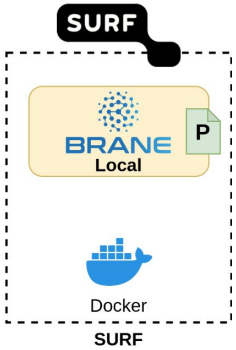
PoC in action - Rosanne's use-case

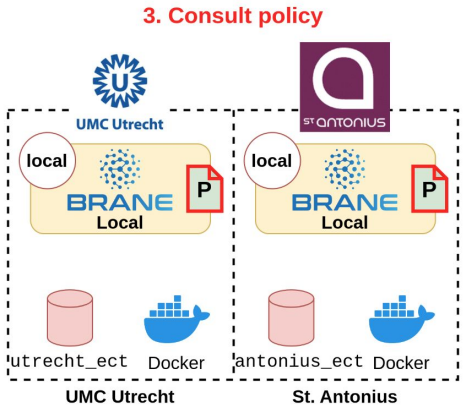


PoC in action - Rosanne's use-case

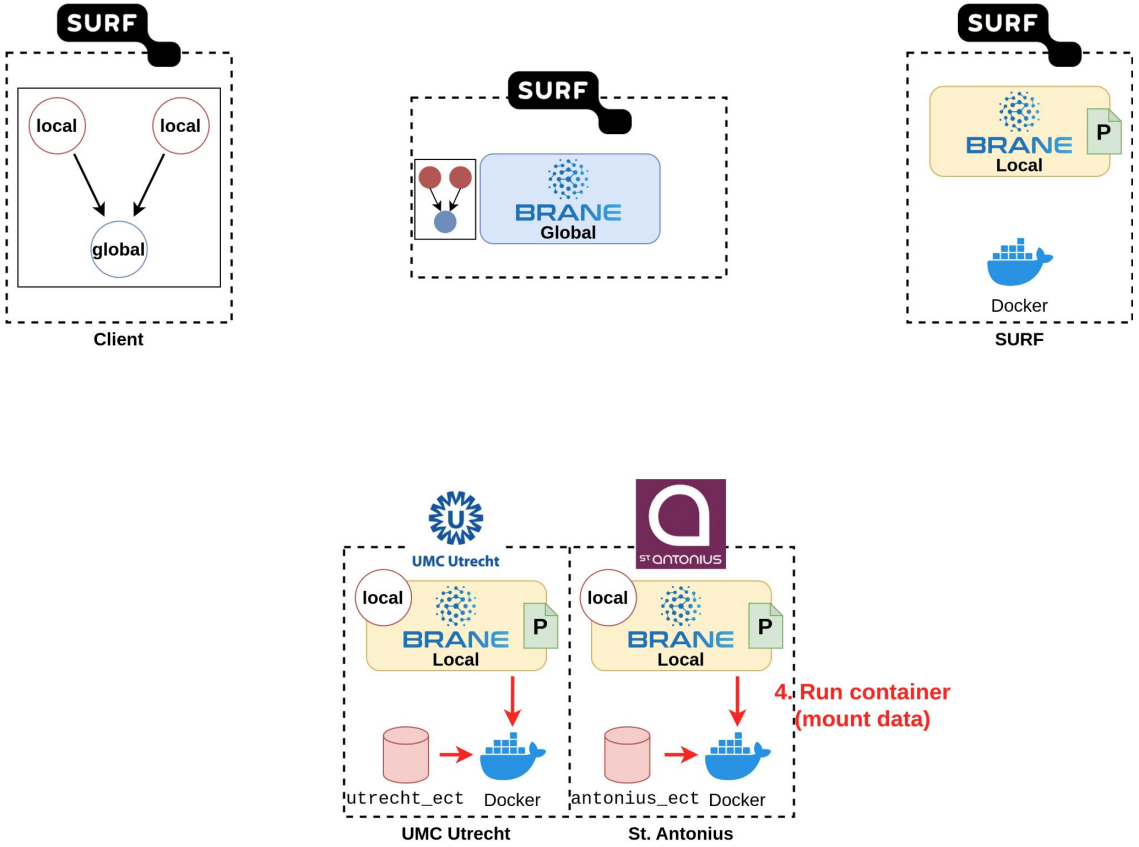




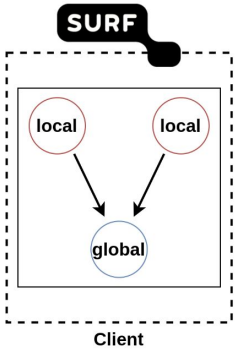


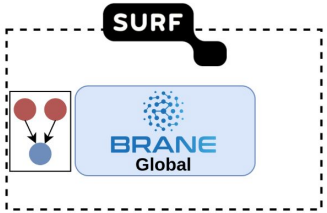


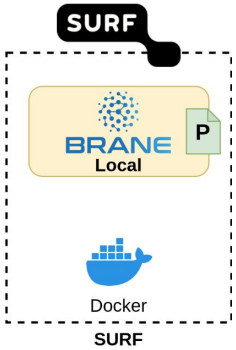
PoC in action - Rosanne's use-case

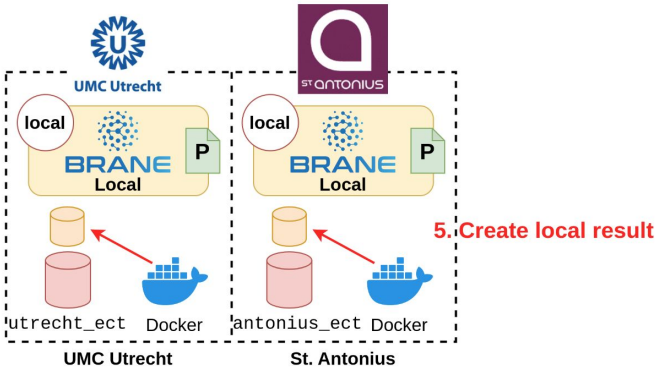


PoC in action - Rosanne's use-case

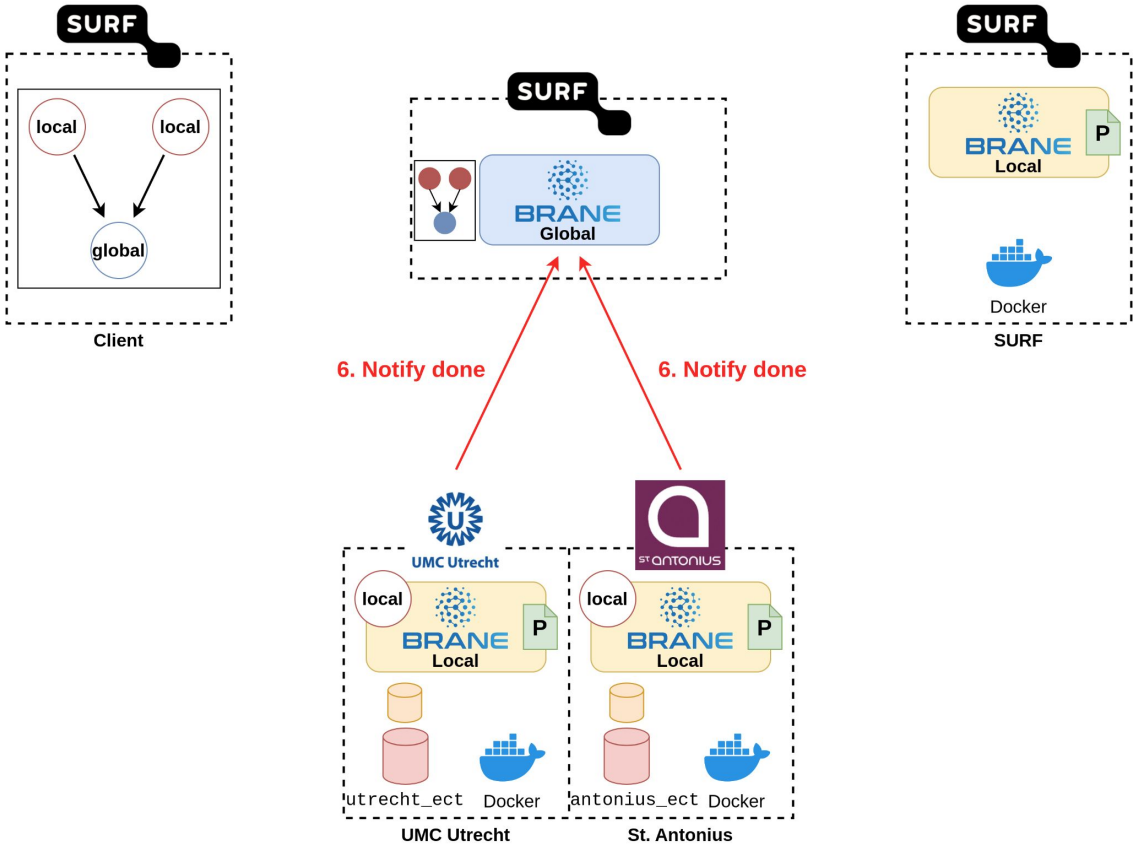




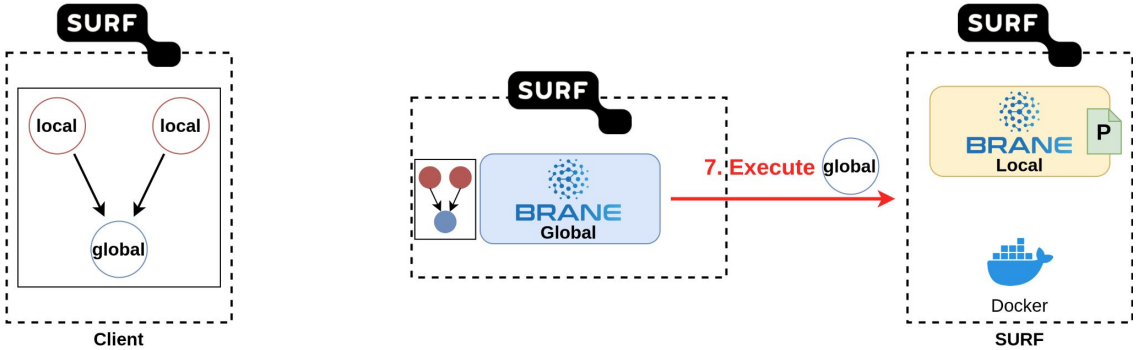


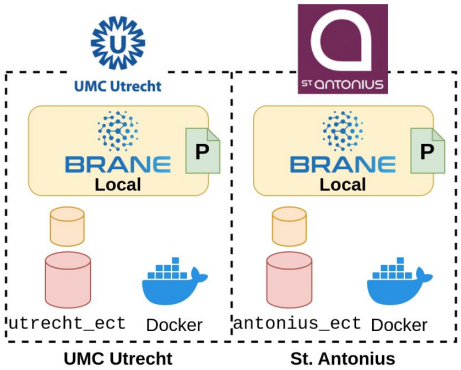


PoC in action - Rosanne's use-case

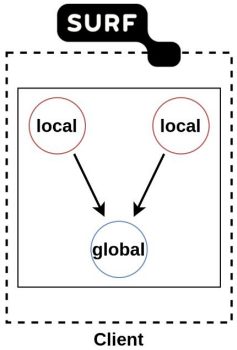


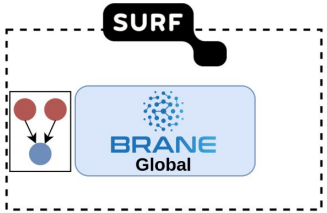
PoC in action - Rosanne's use-case

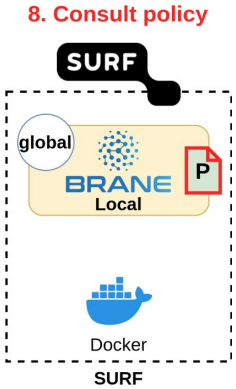


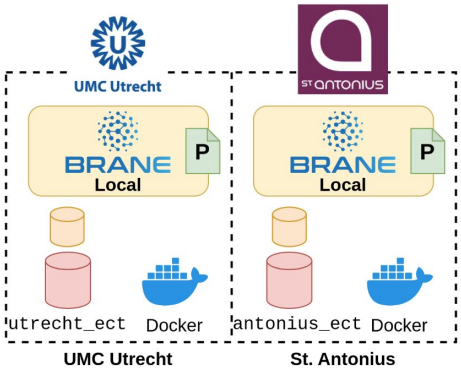


PoC in action - Rosanne's use-case

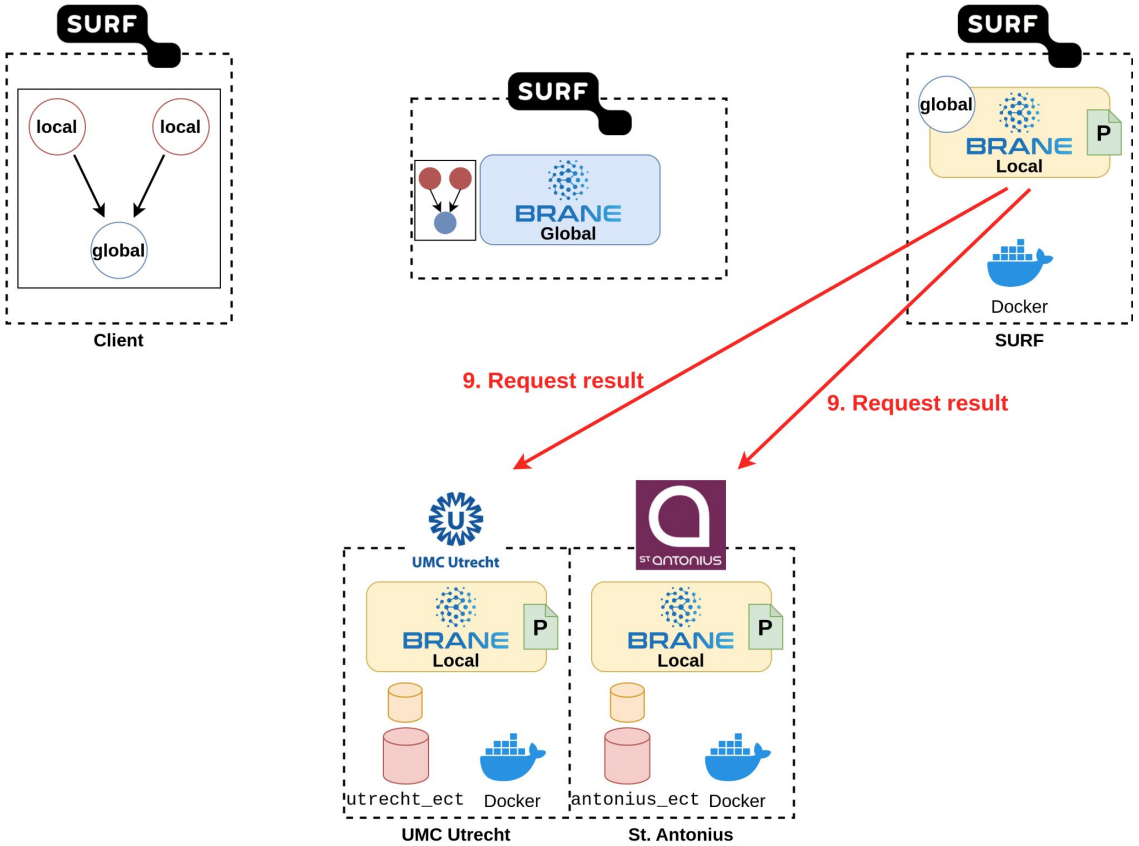




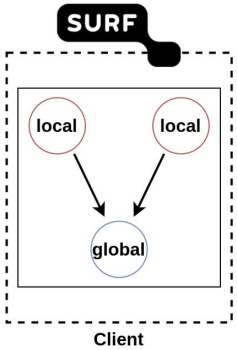


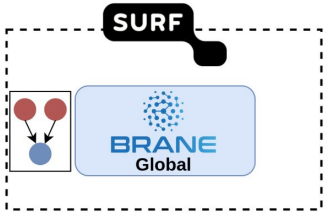


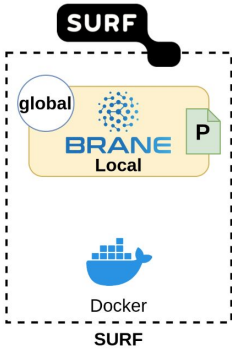
PoC in action - Rosanne's use-case

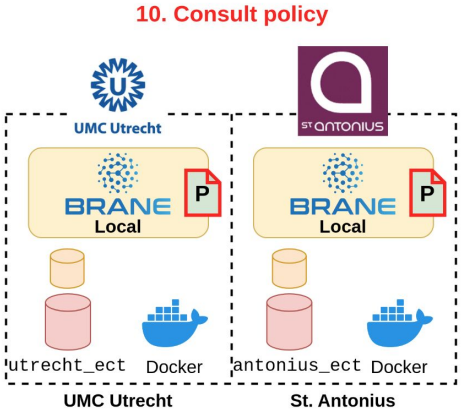


PoC in action - Rosanne's use-case

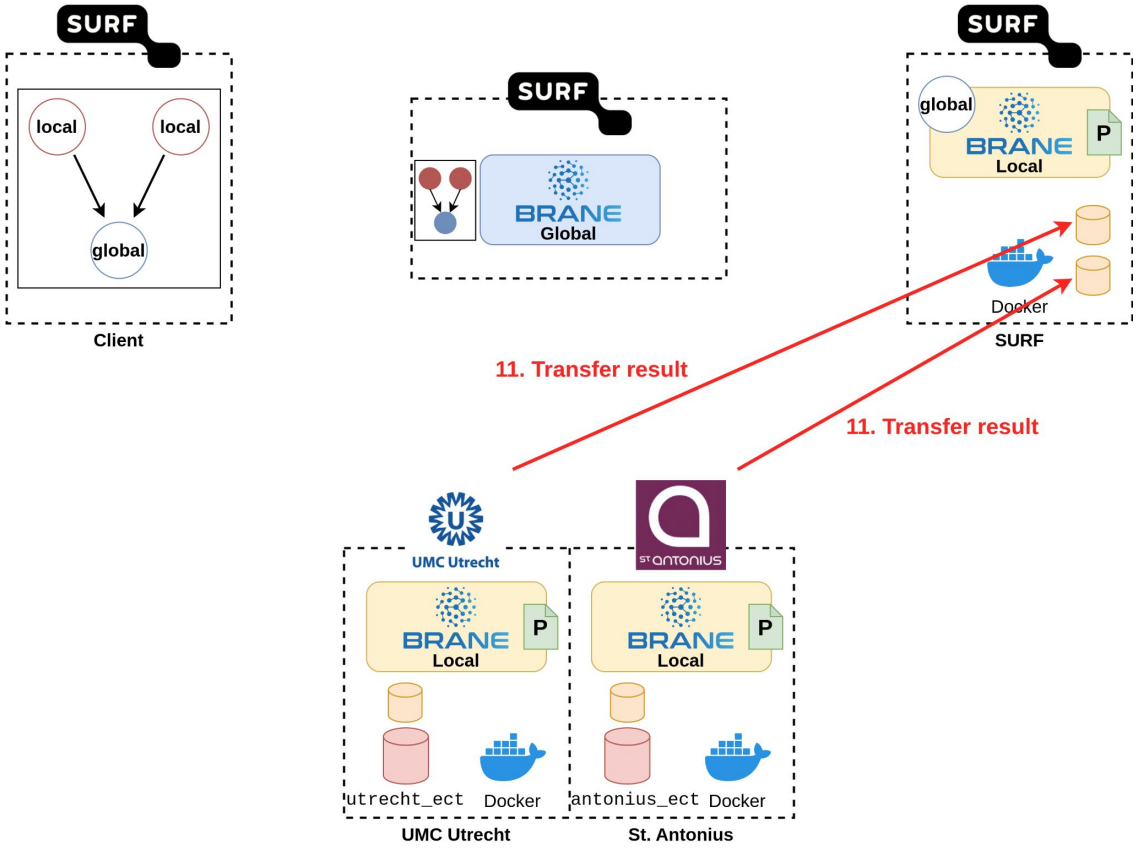




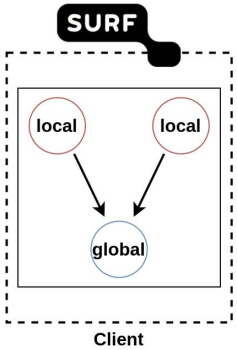


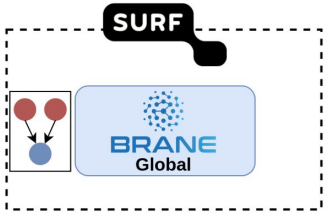


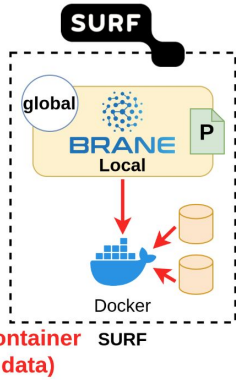
PoC in action - Rosanne's use-case

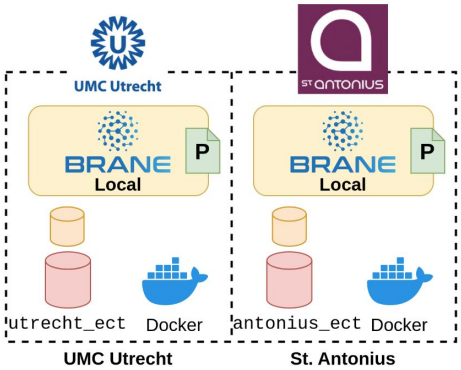


PoC in action - Rosanne's use-case

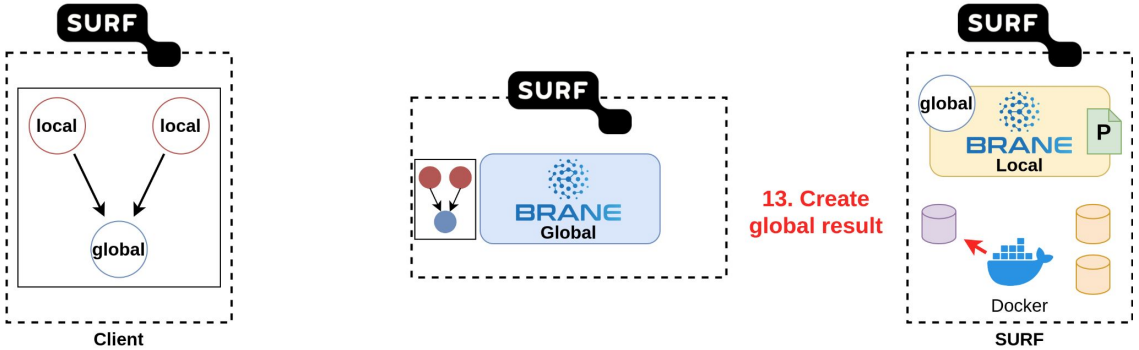


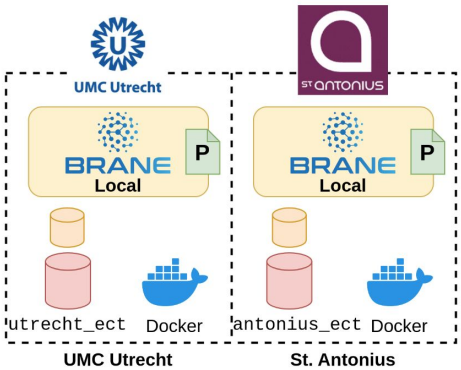




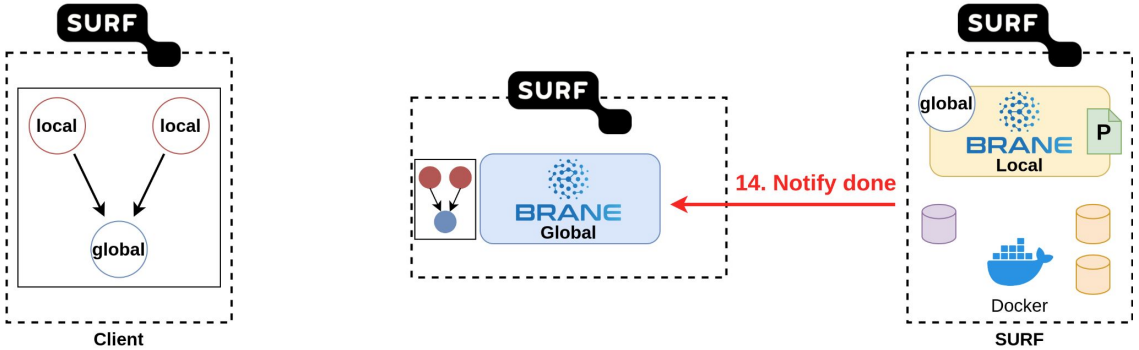


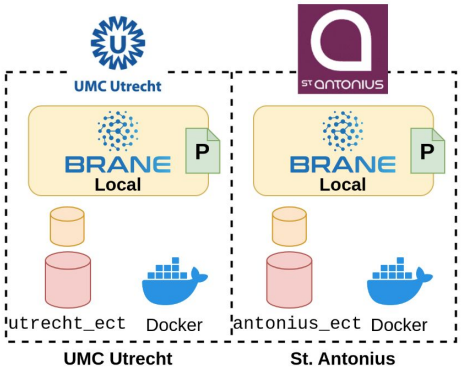
PoC in action - Rosanne's use-case



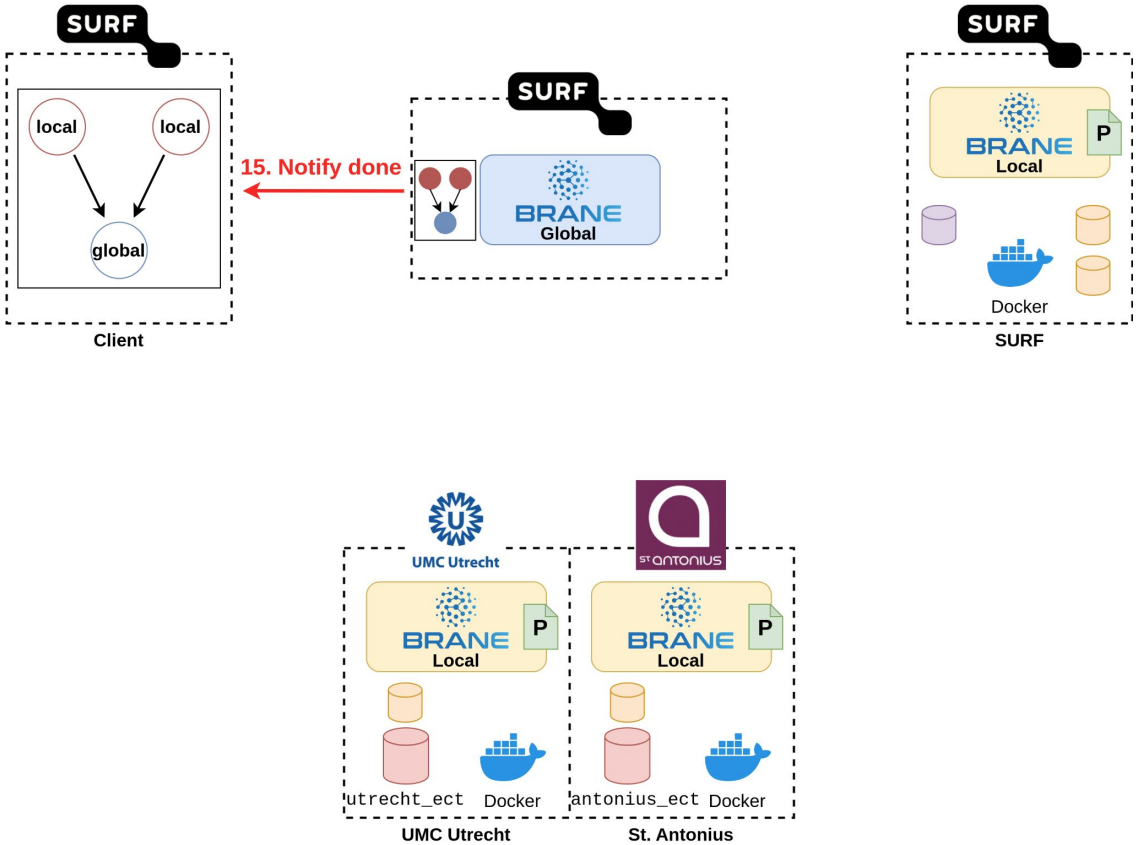


PoC in action - Rosanne's use-case

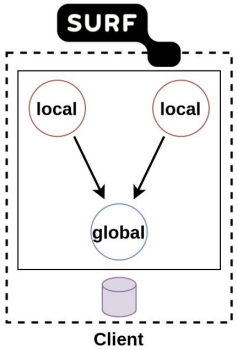


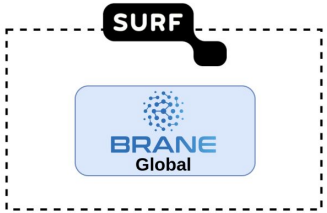


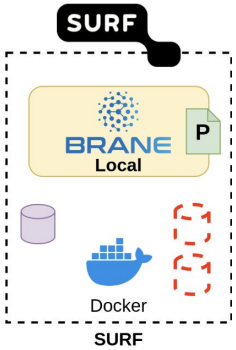
PoC in action - Rosanne's use-case



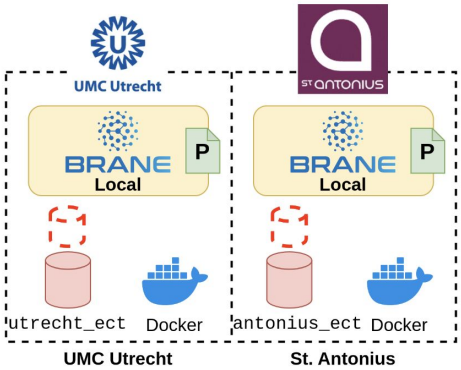
PoC in action - Rosanne's use-case



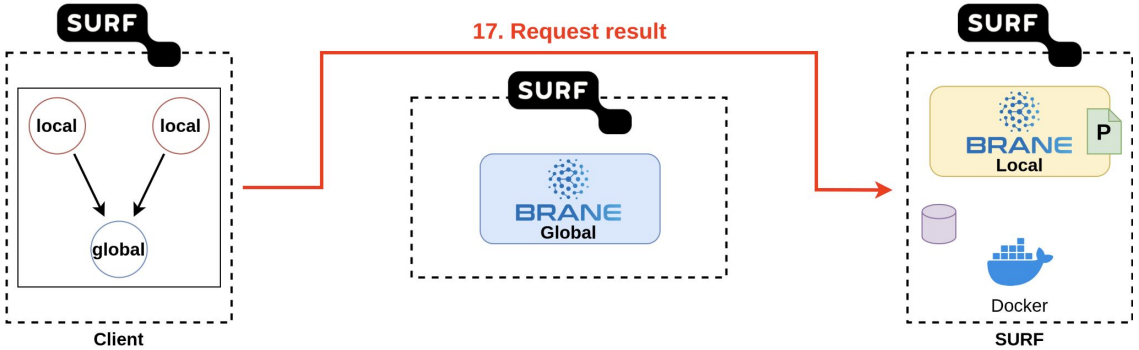


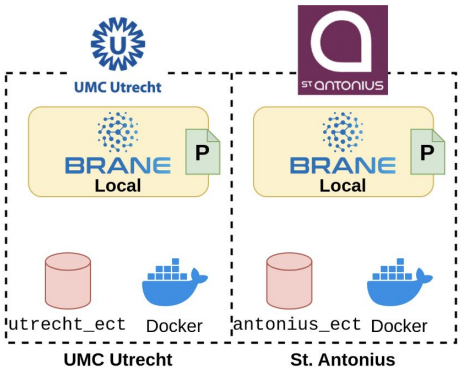


16. Remove intermediate results

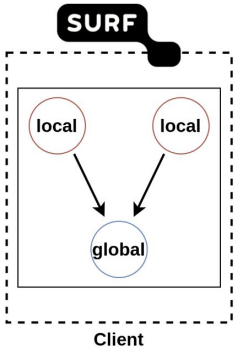


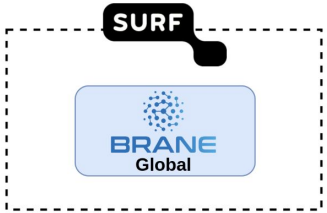
PoC in action - Rosanne's use-case

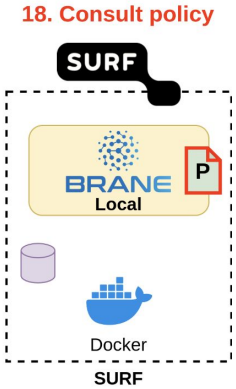


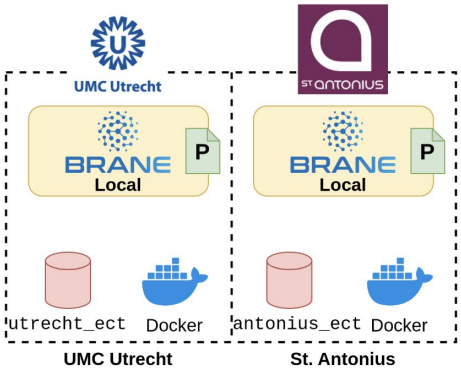


PoC in action - Rosanne’s use-case

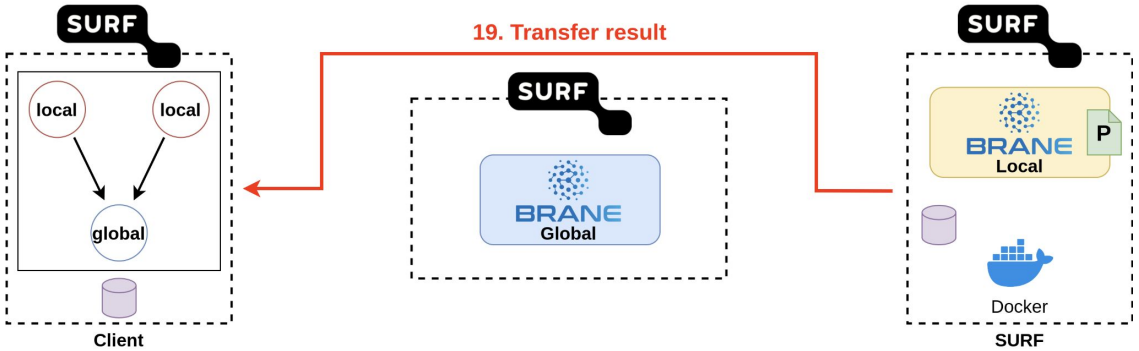


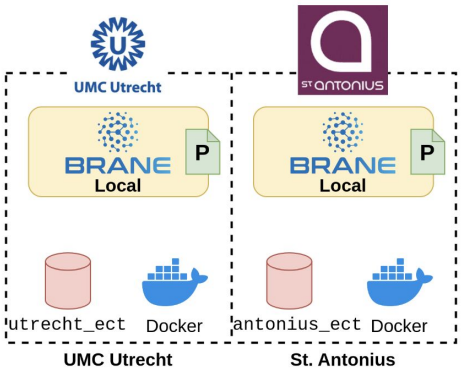




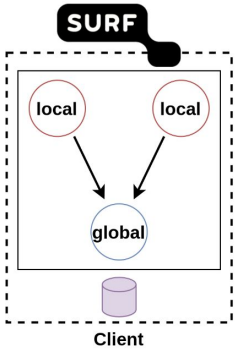


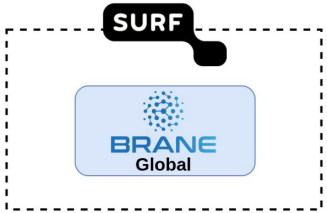
PoC in action - Rosanne's use-case

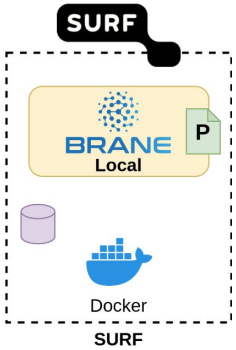




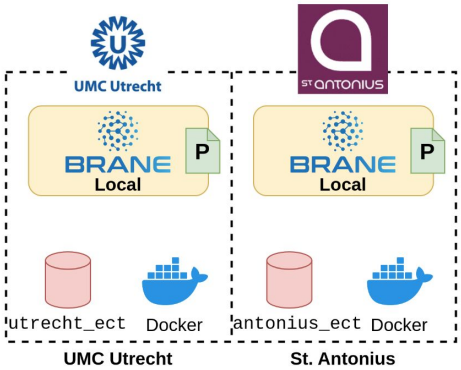
PoC in action - Rosanne's use-case

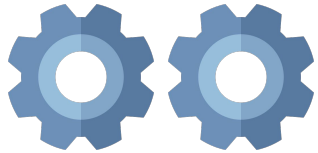






20. Done!!

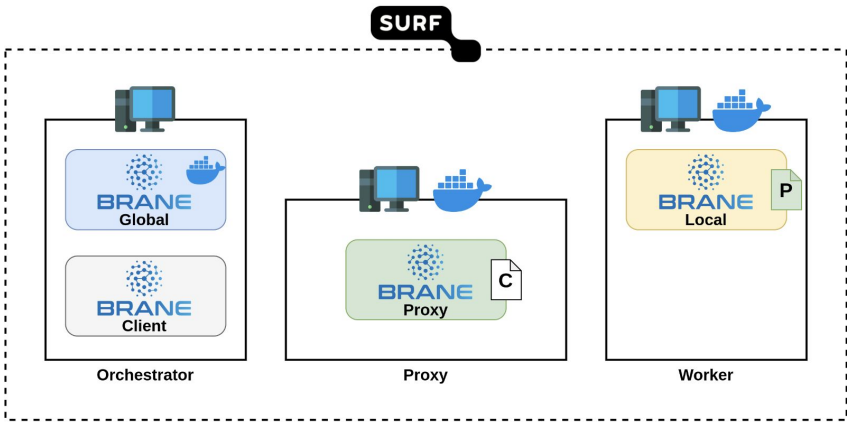


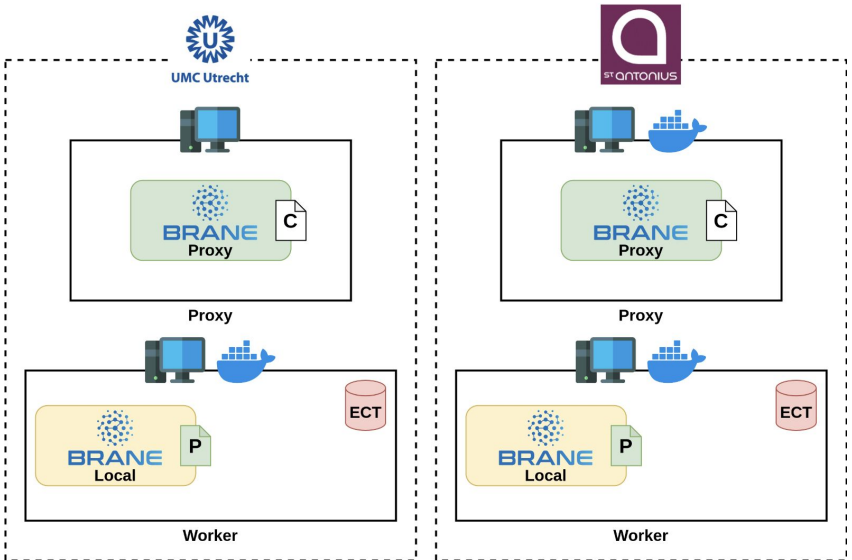


II. PoC Setup

Getting more accurate

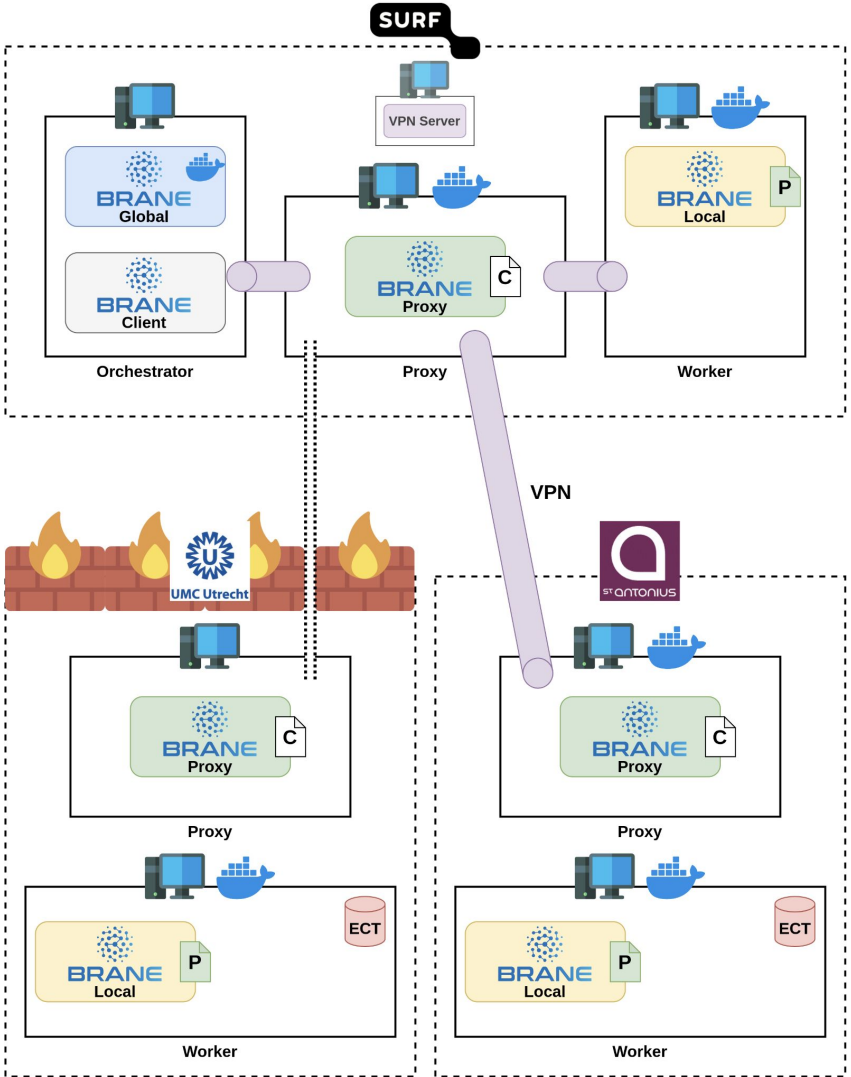
- Let’s get detailed!
- **Multiple VMs** per domain
- **Proxy nodes**
  - Channel communication
  - “Gateways” for nodes
- Third VMs unused
  - No time to add Jamila’s framework





Getting more secure

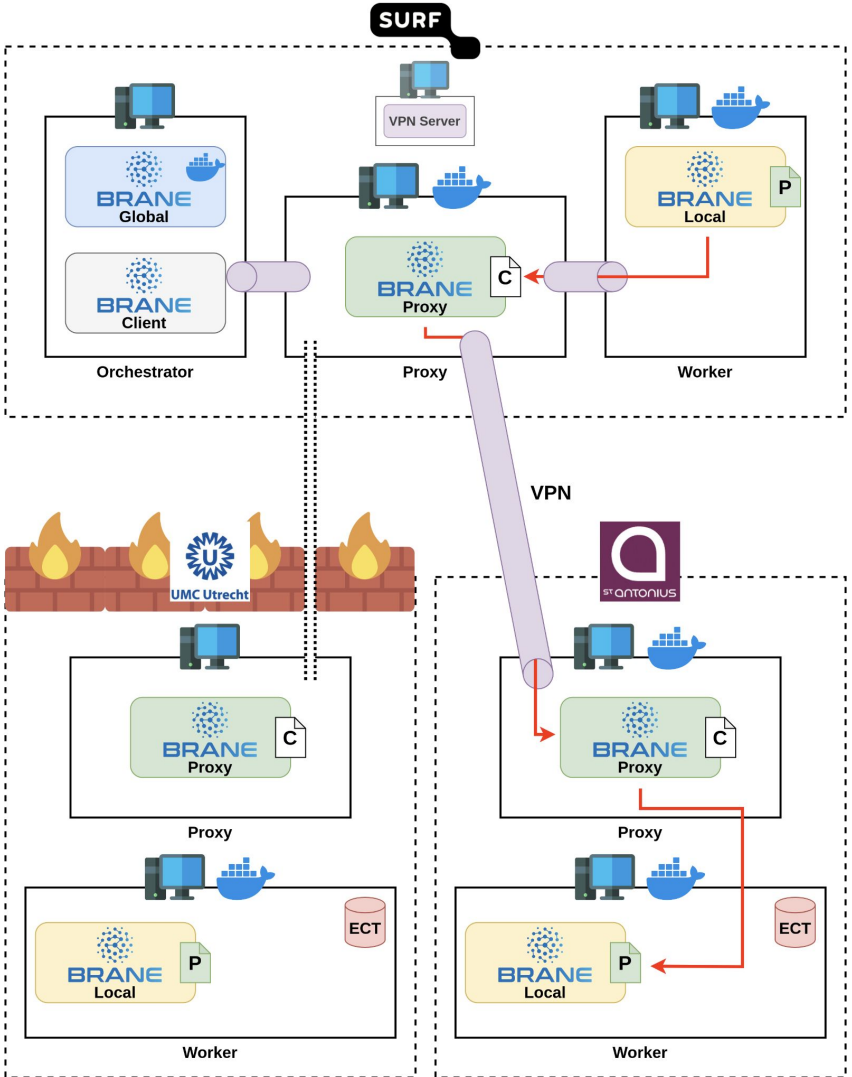
- Attempted to create realistic network
- **VPN** St. Antonius / SURF
  - strongSwan VPN[1] (IPsec)
- **UMC Utrecht firewall** restrictions
  - Only proxy nodes are allowed to talk



[1] https://www.strongswan.org/

Getting more secure

- To adhere to security, we need to define **specific network routes**
  - Specific hops
  - Specific interfaces



III. Configuring Brane nodes

`node.yml`

- Defines node **context**
  - Defines **node kind** (central, worker, proxy, …)
  - Defines **other config locations**
  - Defines **ports**
  - Defines **container names**
  - …
- Comparable to `~/kube/config.yaml`

```yaml
hostnames:
  central: 145.38.187.47
  surf: 145.38.187.47
  umc_utrecht: 143.121.240.12
  st_antonius: 194.13.118.4
node: !worker
  name: umc_utrecht
  paths:
    certs: /home/muller/brane/config/certs
    packages: /home/muller/brane/packages
    backend: /home/muller/brane/config/backend.y
    policies: /home/muller/brane/config/policies
    proxy: null
    data: /home/muller/brane/data
    results: /home/muller/brane/results
    temp_data: /home/muller/brane/temp_data
    temp_results: /home/muller/brane/temp_result
  services:
    prx: !external
      address: http://umc_utrecht:1080
    reg:
      name: brane-reg-umc_utrecht
      address: https://brane-reg-umc_utrecht:108
      bind: 0.0.0.0:1080
      external_address: https://umc_utrecht:1080
    job:
```

```yaml
hostnames:
  central: 145.38.187.47
  surf: 145.38.187.47
  umc_utrecht: 143.121.240.12
  st_antonius: 194.13.118.4
node: !proxy
  paths:
    certs: /home/muller/brane/config/certs
    proxy: /home/muller/brane/config/proxy.yml
  services:
    prx:
      name: brane-prx-proxy
      address: http://brane-prx-proxy:1080
      bind: 0.0.0.0:1080
      external_address: http://proxy:1080
```

`node.yml` - specific interfaces

- We use `hostnames` to **customize addressing**
  - Node-local contents of `/etc/hosts`
- **Different nodes** talk to **different interfaces**
  - …while sending the same hostname around
- Hacky, but it works!

```yaml
hostnames:
  central: 10.0.42.98
  surf: 10.0.42.98
  umc_utrecht: 143.121.240.12
  st_antonius: 194.13.119.49
node: !worker
  name: st_antonius
```

```yaml
hostnames:
  central: 145.38.187.47
  surf: 145.38.187.47
  umc_utrecht: 143.121.240.12
  st_antonius: 194.13.118.4
node: !proxy
  paths:
    certs: /home/muller/brane/config/certs
    proxy: /home/muller/brane/config/proxy.yml
  services:
    prx:
      name: brane-prx-proxy
      address: http://brane-prx-proxy:1080
      bind: 0.0.0.0:1080
      external_address: http://proxy:1080
```

```yaml
hostnames:
  central: 145.38.187.47
  surf: 145.38.187.47
  umc_utrecht: 143.121.240.12
  st_antonius: 194.13.118.4
node: !worker
  name: umc_utrecht
  paths:
    certs: /home/muller/brane/config/certs
    packages: /home/muller/brane/packages
    backend: /home/muller/brane/config/backend.y
    policies: /home/muller/brane/config/policies
    proxy: null
    data: /home/muller/brane/data
    results: /home/muller/brane/results
    temp_data: /home/muller/brane/temp_data
    temp_results: /home/muller/brane/temp_result
  services:
    prx: !external
      address: http://umc_utrecht:1080
    reg:
      name: brane-reg-umc_utrecht
      address: https://brane-reg-umc_utrecht:108
      bind: 0.0.0.0:1080
      external_address: https://umc_utrecht:1080
    job:
```

proxy.yml

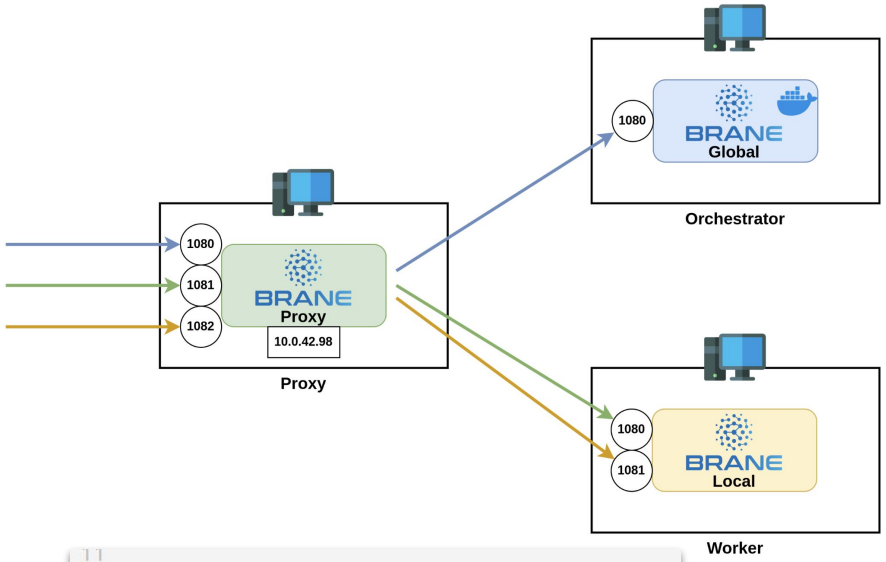
- **Routes** network traffic
- **Authenticates** clients
  - Only clients presenting signed client certificate
- Routes through BFCs

```
[lut_99@gameLinux config]$ branectl generate proxy
Successfully generated ./proxy.yml
[lut_99@gameLinux config]$ _
```

```yaml
outgoing_range:
  start: 1082
  end: 1085
incoming:
  1080: https://143.121.240.11:1080
  1081: https://143.121.240.11:1081
forward: null
```

`proxy.yml` - specific hops

- Route incoming traffic through **single IP**
  - Essentially NATs using the `incoming`-table
- Outgoing traffic already **routed through proxy** by default




```
[lut_99@gameLinux config]$ branectl generate proxy
Successfully generated ./proxy.yml
[lut_99@gameLinux config]$ _
```

```
12
13  outgoing_range:
14    start: 1082
15    end: 1085
16  incoming:
17    1080: https://143.121.240.11:1080
18    1081: https://143.121.240.11:1081
19  forward: null
20
```

policies.yml

- Defines **policies**!
  - Implemented as simple rule-based rules
- One set defines **which container to execute**
- One set defines **who can access which dataset**
  - Identification based on **client-side certificates**

```
[lut_99@gameLinux worker]$ branectl packages hash ~/.local/share/brane/packages_2.0.0/epi_rosanne/1.0.0/image.tar
QS43h4ycr/PdYZTwUAKwOc68qKEZiz9oDWCo0kMdgGE=
[lut_99@gameLinux worker]$ _
```

```yaml

users:
- policy: deny
  user: surf
  data: umc_utrecht_ect
- policy: deny
  user: surf
  data: umc_utrecht_ect_train
- policy: deny
  user: surf
  data: umc_utrecht_ect_test
- policy: allow_user_all
  user: surf
- policy: deny
  user: client
  data: umc_utrecht_ect
- policy: deny
  user: client
  data: umc_utrecht_ect_train
- policy: deny
  user: client
  data: umc_utrecht_ect_test
- policy: allow_user_all
  user: client
- policy: deny_all
containers:
- policy: allow
  name: Rosanne's use-case container
  hash: "QS43h4ycr/PdYZTwUAKwOc68qKEZiz9oDWCo0kMd
- policy: allow
  name: Saba train synthetic container
  hash: "HdyBrlwCpL9ltQDl7u9Ac/vexBBInRgoWSuHR09
- policy: deny_all

```

Certificates

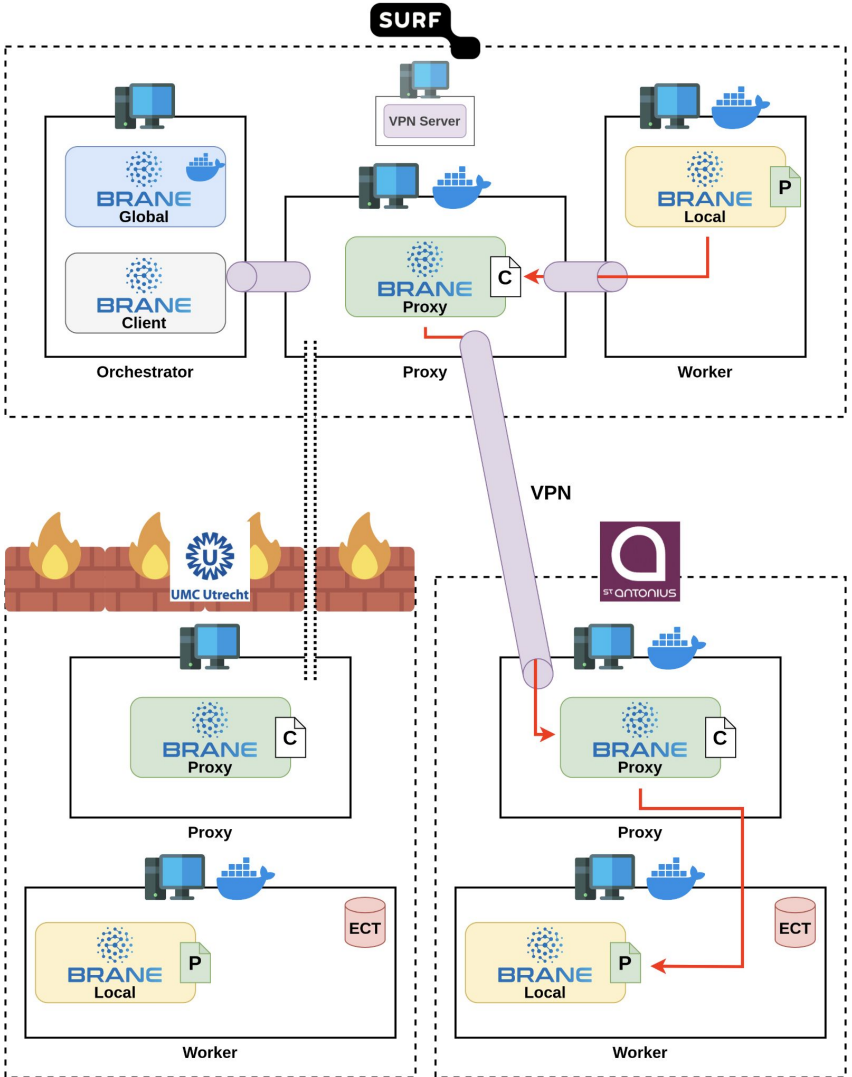
- **Authenticates clients**
  - So that policy may authorize them
- Used to **encrypt traffic** (without BFCs, that is)
- Required:
  - Root certificate (per node)
  - Server certificate (per node)
  - Client certificate (per node, per client)

```
[lut_99@gameLinux certs]$ branectl generate certs server umc_utrecht
Downloading http://github.com/cloudflare/cfssl/releases/download/v1.6.3/cfssl_1.6.3_linux_amd64...
 > Checksum 16b42bfc592dc4d0ba1e51304f466cae7257edec13743384caf4106195ab6047 OK
Successfully generated server certificates for domain umc_utrecht
[lut_99@gameLinux certs]$ ls
ca.csr  ca-key.pem  ca.pem  server.csr  server-key.pem  server.pem
[lut_99@gameLinux certs]$ _
```

```
[lut_99@gameLinux surf]$ branectl generate certs client surf -c ../../ca.pem -k ../../ca-key.pem
Successfully generated client certificates for domain surf
[lut_99@gameLinux surf]$ _
```

Takeaways

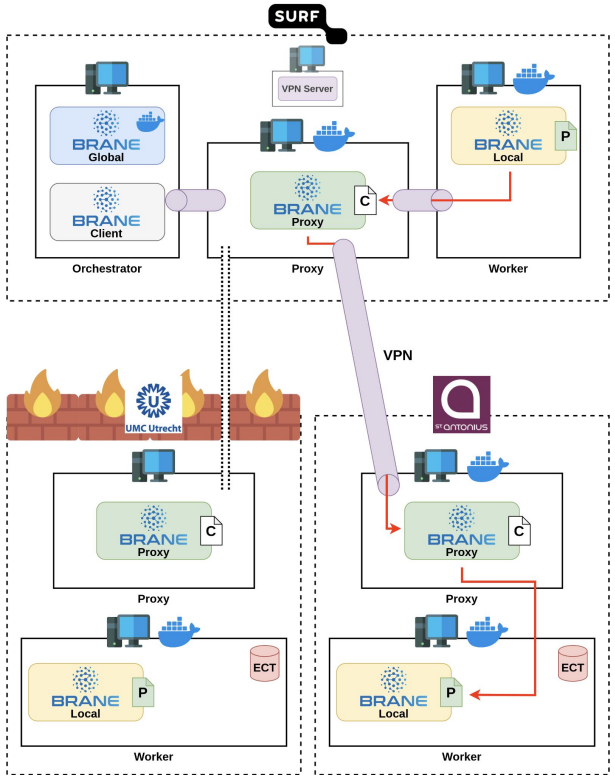
- Brane allows configuring **specific network routes**
  - Specific hops
  - Specific interfaces
- Policies defined through **rule-based YAML file**
- **Certificates** used for authentication/encryption



IV. Conclusion

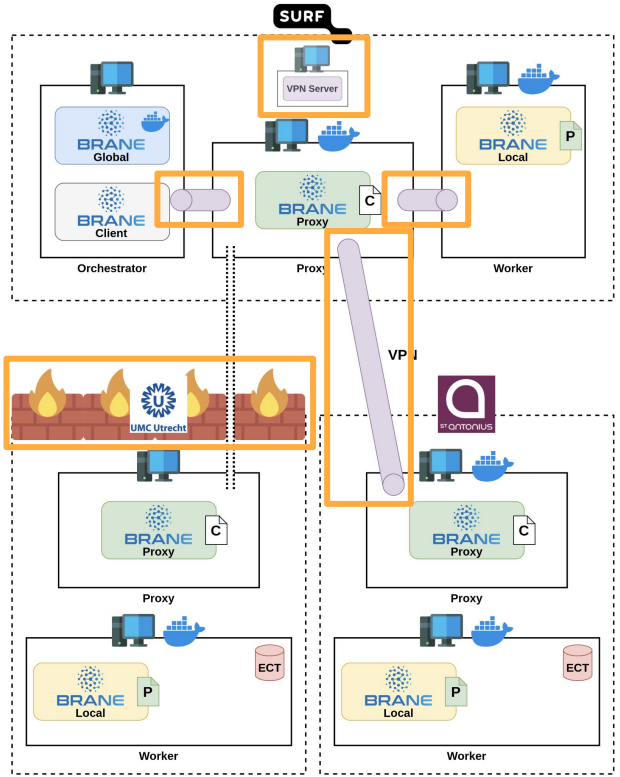
EPIF in the PoC

- Setup between **SURF**, **UMC Utrecht** and **St. Antonius**
- Two **use-cases**
  - **Rosanne's** stratified confidence sequence analysis
  - **Saba's** synthetic data generation
- Realistic **network security** (hopefully)
  - Brane supports required routing requirements
    - …also *because* of the Proof-of-Concept



What if… BFC Framework

- BFC Framework can add in security as **Virtualized Network Functions**
  - Spawn as Docker container
  - Route traffic through container
- Can interact with **policy**
  - e.g., *“Only share with St. Bob Hospital if they are trusted and setup a VPN with us”*
- Only useful for **inter-domain** networking



BRANE

Tim Müller (t.muller@uva.nl)

https://enablingpersonalizedinterventions.nl

https://github.com/epi-project/brane

https://wiki.enablingpersonalizedinterventions.nl

The icons (not logos) in this presentation are from: Freepik, juicy_fish, Ultimatearm, Vector Valley