M M S
EXPERIENCE
BEYOND
DIGITAL

December 2022

Confidential Computing: Hands-On

Marvin Wolf
IT Security
Consultant

Ivan Gudymenko
IT Security
Architect

T Systems

Agenda

- **Motivation**
- **Principles**
- **Workflow**
- **Implementation**
- **Examples**

Motivation

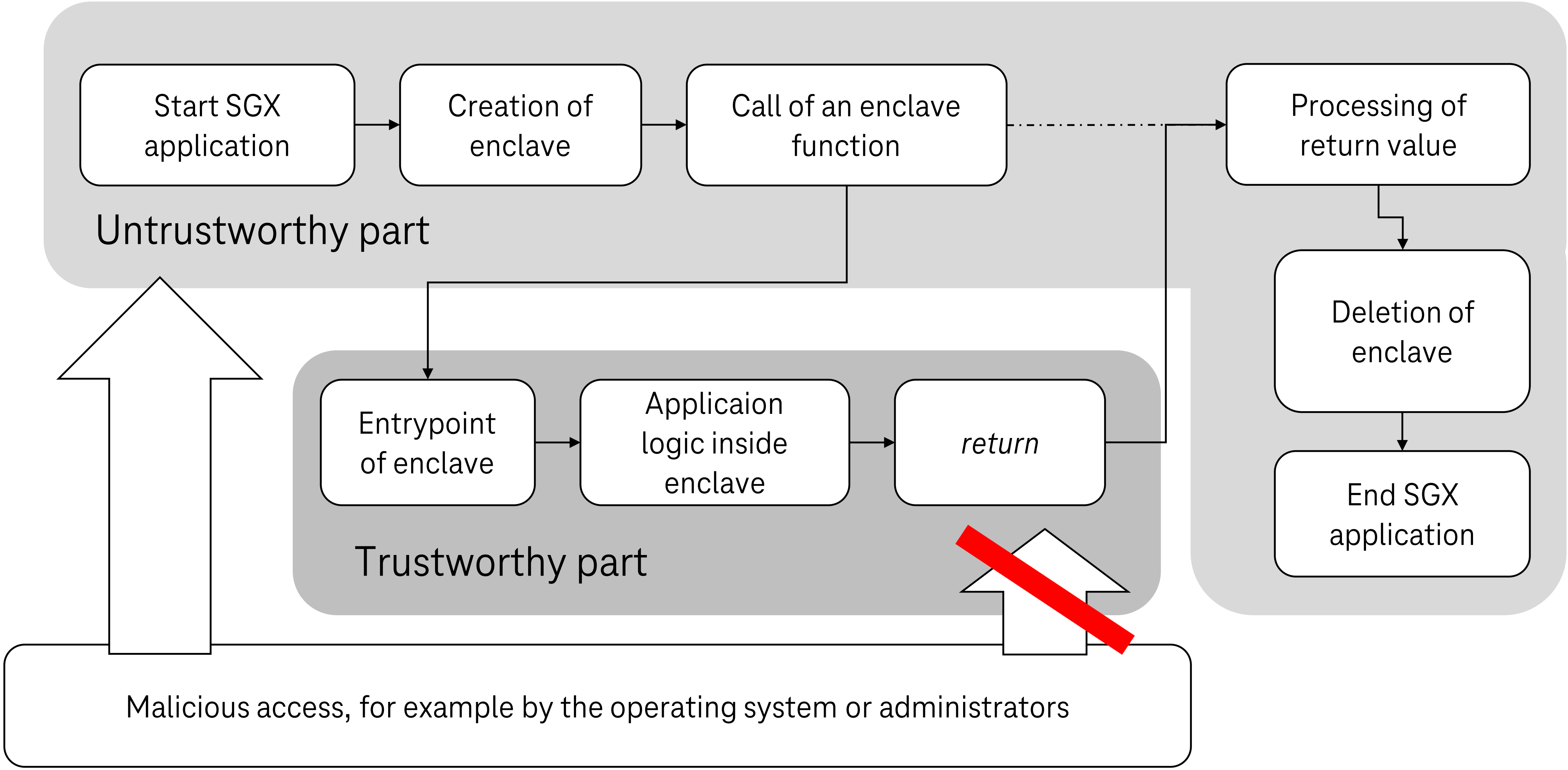
- **Increasing use of cloud computing**
- **Data in transit and data at rest can be secured**
- **Data in use, in cleartext in memory**
- **Threats:**
  - Internal attacker/ administrator
  - External attacker
  - Confidentiality and integrity
- **Privacy protecting compliant processing**

Principles

- **Workload oriented: Intel SGX**
- Instruction set extension
- Separation of the application into two parts
- → isolation of the trustworthy code
- Exclusive memory areas, which encrypted during runtime

- **System oriented: AMD SEV-SNP**
- Extension of processors
- Encryption of entire main memory during runtime
- Entire machine including virtual machines are protected

Workflow


Start SGX application
Creation of enclave
Call of an enclave function
Processing of return value
Untrustworthy part
Deletion of enclave
End SGX application
Entrypoint of enclave
Applicaion logic inside enclave
return
Trustworthy part
Malicious access, for example by the operating system or administrators

Implementation

- **Focus on separation during development**

- **Existing applications need to be migrated**
  - Intel SGX SDK (C/C++)
  - Open Enclave SDK (C/C++)
  - EGo framework (Go)
  - Scone (Java, Python, Rust, Docker container)
  - Anjuna (Docker container)
  - Library OS's: Gramine, Occlum

Examples

- **Hello-World server**
- **Embedding files**
- **Remote attestation**
- **Sconification**
- **Real world example: Vault**

Hello-World Server

```
package main

import (
      "fmt"
      "net/http"
)
func main() {
   http.HandleFunc("/", HelloServer)
   http.ListenAndServe(":8080", nil)
}

func HelloServer(w http.ResponseWriter, r *http.Request) {
   fmt.Fprintf(w, "Hello, world!\n")
   fmt.Fprintf(w, "This is a test!")
}
```

1. ego-go build // build application
2. ego sign helloworld // sign application
3. ego run helloworld // execute application

enclave.json

Enclave‘s configuration defined in JSON

Applied when signing the executable (ego sign)

```
{
 "exe": "helloworld",                  // path to the executable
 "key": "private.pem",                 // path to private RSA of the signer
 "debug": true,                        // debuggable
 "heapSize": 512,                      // heap size available for the enclave
 "executableHeap": false,              // if true, the enclave heap will be executable, for libraries that JIT-compile code
 "productID": 1,                       // assigned by the developer, so attester can differentiate between different enclaves
 "securityVersion": 1,                 // increased when a security fix is applied to enclave code
 "mounts": null,                       // defining custom mount points for the file system inside the enclave
 "env": null,                          // setting environment variables or getting from the host
 "files": null                         // specifies files that will be embedded, included in enclave measurement
}
```

Source: https://docs.edgeless.systems/ego/reference/config

Embedding files

```
package main

import (
	"fmt"
	"net/http"
)

func main() {
	fmt.Println("Getting https://www.t-systems-mms.com/")
	resp, err := http.Get("https://www.t-systems-mms.com/")
	if err != nil {
		panic(err)
	}
	fmt.Println(resp.Status)
}
```

Host CA-certificates not available inside the enclave

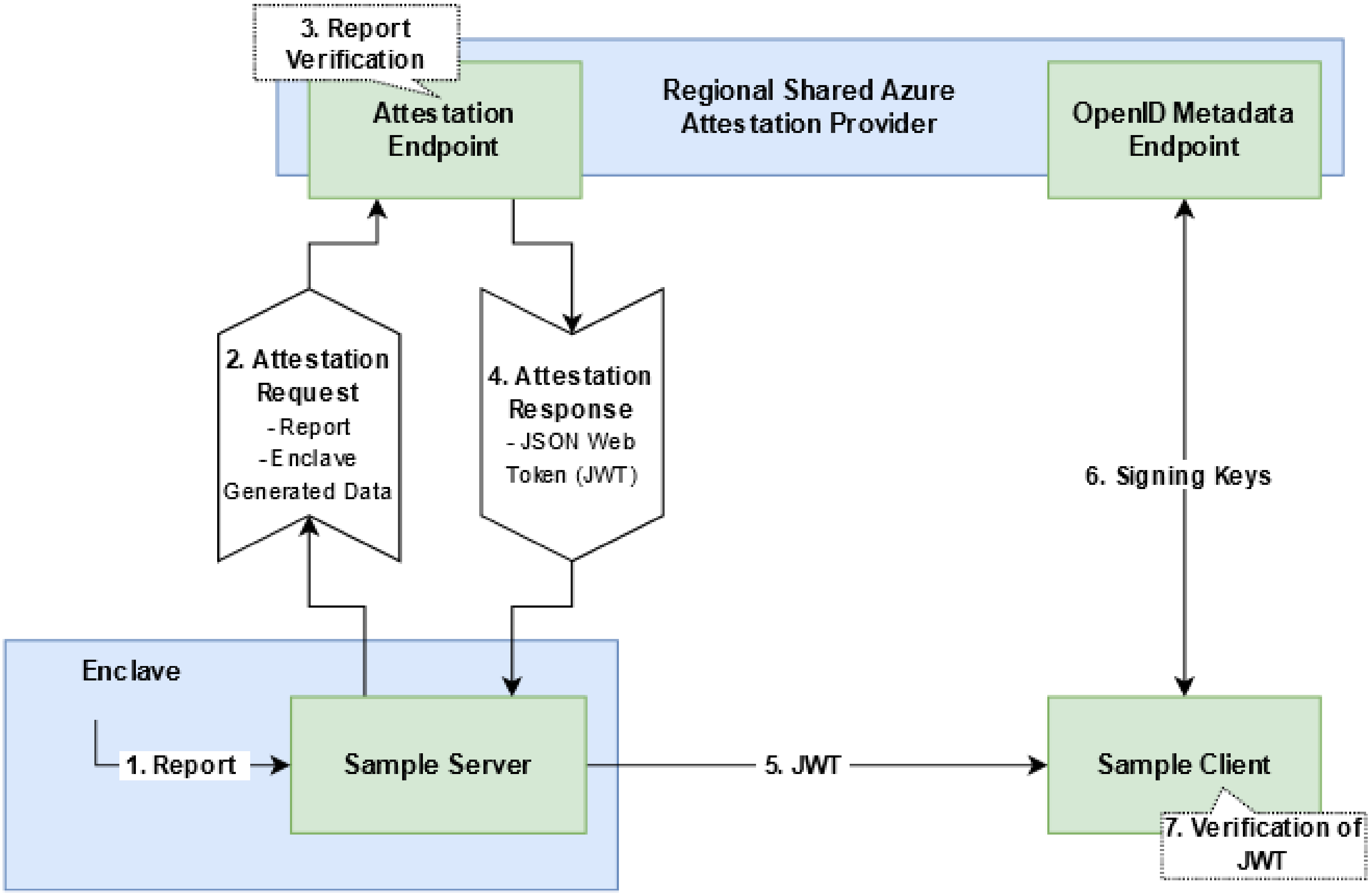
→ security risk

enclave.json

```
{
  "exe": "embedded_file",
  "key": "private.pem",
  "debug": true,
  "heapSize": 512,
  "productID": 1,
  "securityVersion": 1,
  "files": [                                                  // mapping host files inside the enclave, included in the enclave measurement
    {                                                         // can't be manipulated, accessible at runtime via in-enclave-memory
      "source": "/etc/ssl/certs/ca-certificates.crt",// file which should be embedded
      "target": "/etc/ssl/certs/ca-certificates.crt“  // path where file reside at runtime
    }
  ]
}
```

Source: https://docs.edgeless.systems/ego/knowledge/tls

Remote attestation



Source: https://github.com/edgelesssys/ego/tree/master/samples/azure_attestation

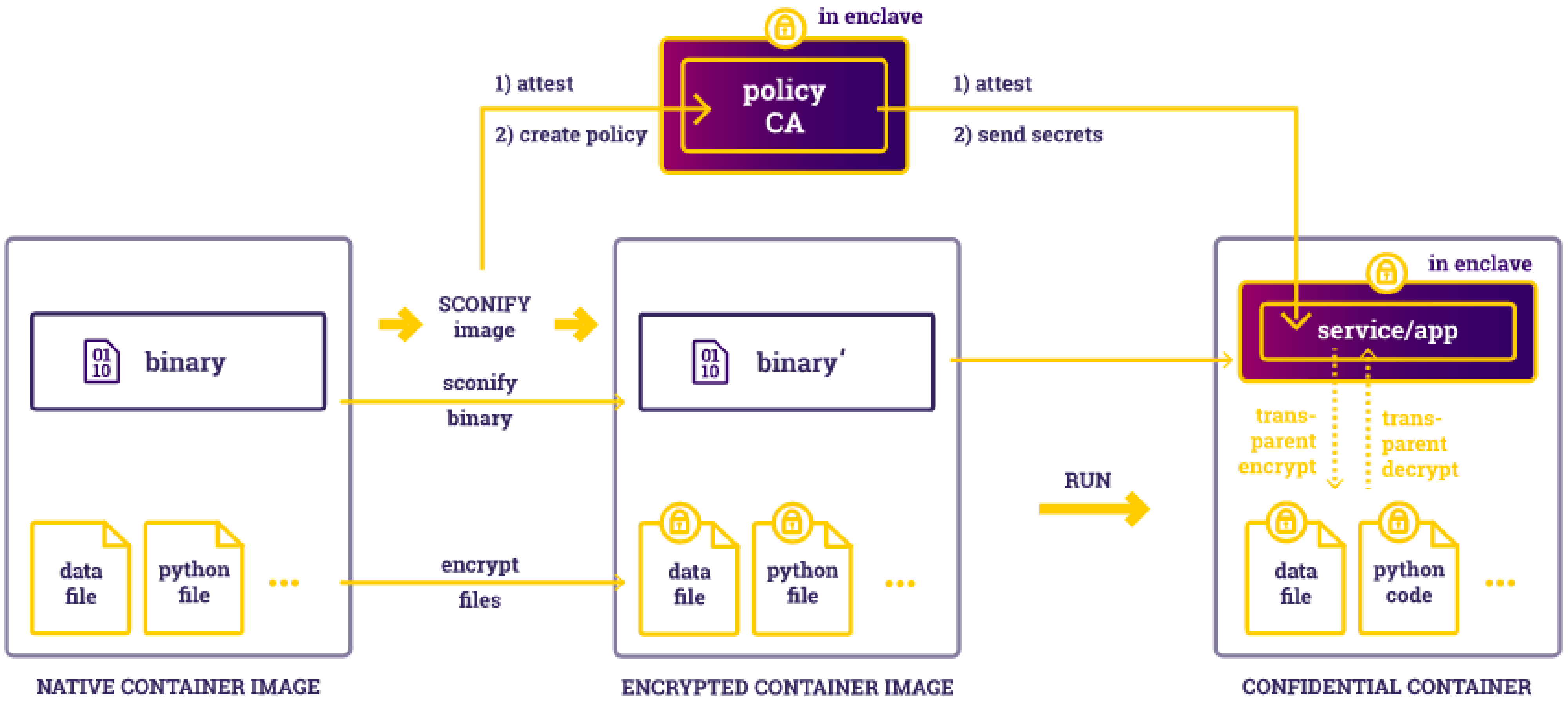
```
ego-go build

ego sign server

ego run server                    // generates report of enclave and verifies it with azure, receives back the attestation token


go build ra_client/client.go      // builds client
```

Client needs signerID (MRSIGNER) as argument to verify the enclave. The ID can be derived from the public key of the signer, by ego signerid.

```
./client -s `ego signerid public.pem`      // gets token from server and public key from attestation provider, verifies both
```

→ establish secured TLS connection and send secret

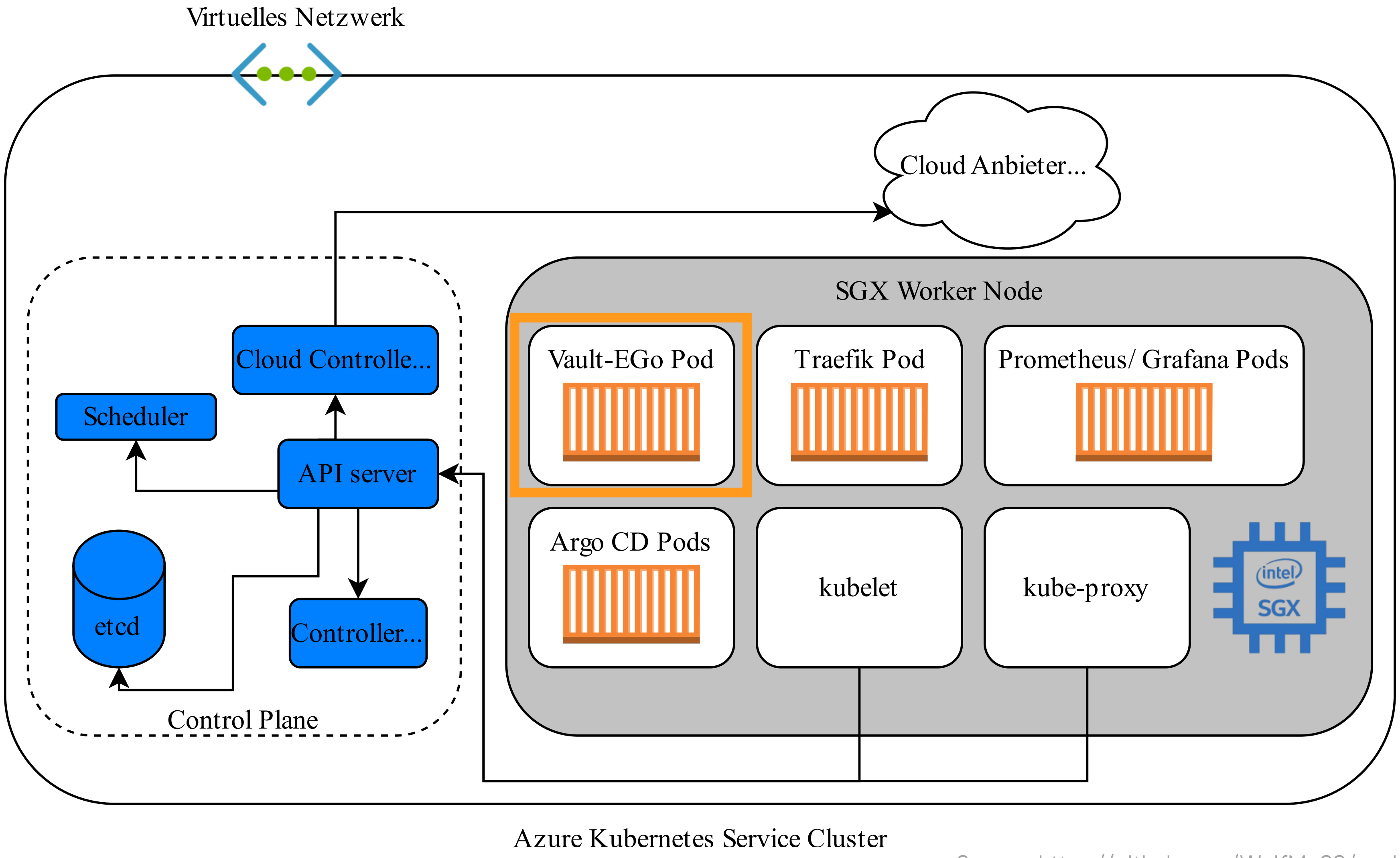
Sconification: lift and shift



Source: https://sconedocs.github.io/ee_sconify_image/

```
CAS_ADDR="scone-cas.cf"
LAS_ADDR="localhost:18766“
SERVICE="my-flask-service“
SESSION="my-flask-session“
NAMESPACE="my-workflow-namespace-$RANDOM“
SCONE_HEAP="1G"
SCONE_STACK="4M"
SCONE_ALLOW_DLOPEN="2“

BINARY="/usr/bin/python3.7„

NATIVE_IMAGE="my-repo/my-native-flask"
ENCRYPTED_IMAGE="my-repo/my-encrypted-flask"

docker run --rm --device="$SGXDEVICE" \
  -v /var/run/docker.sock:/var/run/docker.sock \
  registry.scontain.com/sconecuratedimages/sconecli:sconify-image \
  sconify_image --from="$NATIVE_IMAGE" --to="$ENCRYPTED_IMAGE" --cas-debug \
  --binary="$BINARY" \
  --cas="$CAS_ADDR" --las="$LAS_ADDR" \
  --service-name="$SERVICE" --name="$SESSION" --namespace="$NAMESPACE" \
  --create-namespace \
  --heap="$SCONE_HEAP" --stack="$SCONE_STACK" --dlopen="$SCONE_ALLOW_DLOPEN"
```

Real world example: Vault



Source: https://github.com/WolfMa99/vault-confidential-computing

- **HashiCorp Vault Binary compiled with EGo**
- **Binary and needed resources bundled into Docker image**
- **Docker image used for Kubernetes Pod**
- **Kubernetes cluster has SGX capability**

- **EGo Vault Binary uses SGX capability during runtime**
- **Vault is secured during runtime**

- **Connections to Vault are secured by Let's Encrypt certificates**
- **For demo purposes data is stored inside the EGo Vault container**

- **Data is secured during all three states**

Conclusion

- Different technical solutions for confidential computing
- Different implementations for new and existing applications

- Frameworks and Library OS’s support the migration

Sources

- Costan, V./Lebedev, I./Devadas, S. (2017): *Secure Processors Part I: Background, Taxonomy for Secure Enclaves and Intel SGX Architecture*, in: Jongh, M. (Hrsg.), *Foundations and Trends in Electronic Design Automation: Vol. 11: No. 1-2*, Boston.
- Costan, V./Devadas, S. (2016): *Intel SGX Explained*, in: *IACR Cryptol. ePrint Arch* 1, S. 1–118.
- Arnautov, S. et al. (2016): *SCONE: Secure Linux Containers with Intel SGX*, in: Association, U. (Hrsg.), *12th USENIX Symposium on Operating Systems Design and Implementation*, Savannah.
- https://learn.microsoft.com/de-de/azure/confidential-computing/confidential-containers-enclaves
- https://github.com/edgelesssys/ego