Building resilient systems inside the mesh: abstraction and automation of Virtual Service generation

Vladimir Georgiev, Thought Machine

Sync calls failures inside the mesh

- Everyone says to fail fast and retry quickly, but...
- How fast to timeout?
  - If it's too early then failed the request for no reason.
  - If it's too late then the calling client might be left hanging for too long.
- What errors are retryable?
- Who knows the answer to all the questions?
- How to implement this to be language agnostic?

Virtual Services API

- Solves our problems, but...
- All Service Owners must be aware of the Virtual Services API in order to define their SLOs.
- Potential typing errors when dealing with YAMLs.
- Potential drift between the state of the service API and the Virtual Service config.
- Hard to manage when having hundreds of services.

Abstracting to proto files

```proto
syntax = "proto3";

package sla;
import "google/protobuf/descriptor.proto";

extend google.protobuf.MethodOptions {
    // Describes the anticipated SLOs for the RPC and can override the global
    // Virtual Service Config. See SLO for more description.
    SLO endpoint_slo = 1;
}

message EndpointSLO {
    // Defines the global route timeout.
    string timeout = 1;
    // Defines the upstream timeout per attempt.
    string per_try_timeout = 2;
    // Defines retry attempts amount.
    int32 retries = 3;
    // Defines the retriable gRPC errors.
    repeated RetryOn retry_on = 4;
}

enum RetryOn {
    CANCELLED = 0;
    DEADLINE_EXCEEDED = 1;
    INTERNAL = 2;
    RESOURCE_EXHAUSTED = 3;
    UNAVAILABLE = 4;
}
```

Annotations API definition

```proto
syntax = "proto3";

package hello_world;

import "demo/virtual_service/sla.proto";

service Greeting {
    rpc SayHi(GreetingRequest) returns (GreetingResponse) {
        option (virtual_service.endpoint_slo) = {
            timeout: "200ms"
            per_try_timeout: "150ms"
            retries: 5
        };
    };
};

message GreetingRequest {
    string msg = 1;
}

message GreetingResponse {
    string msg = 1;
}
```

Greeting service example

Please Build System

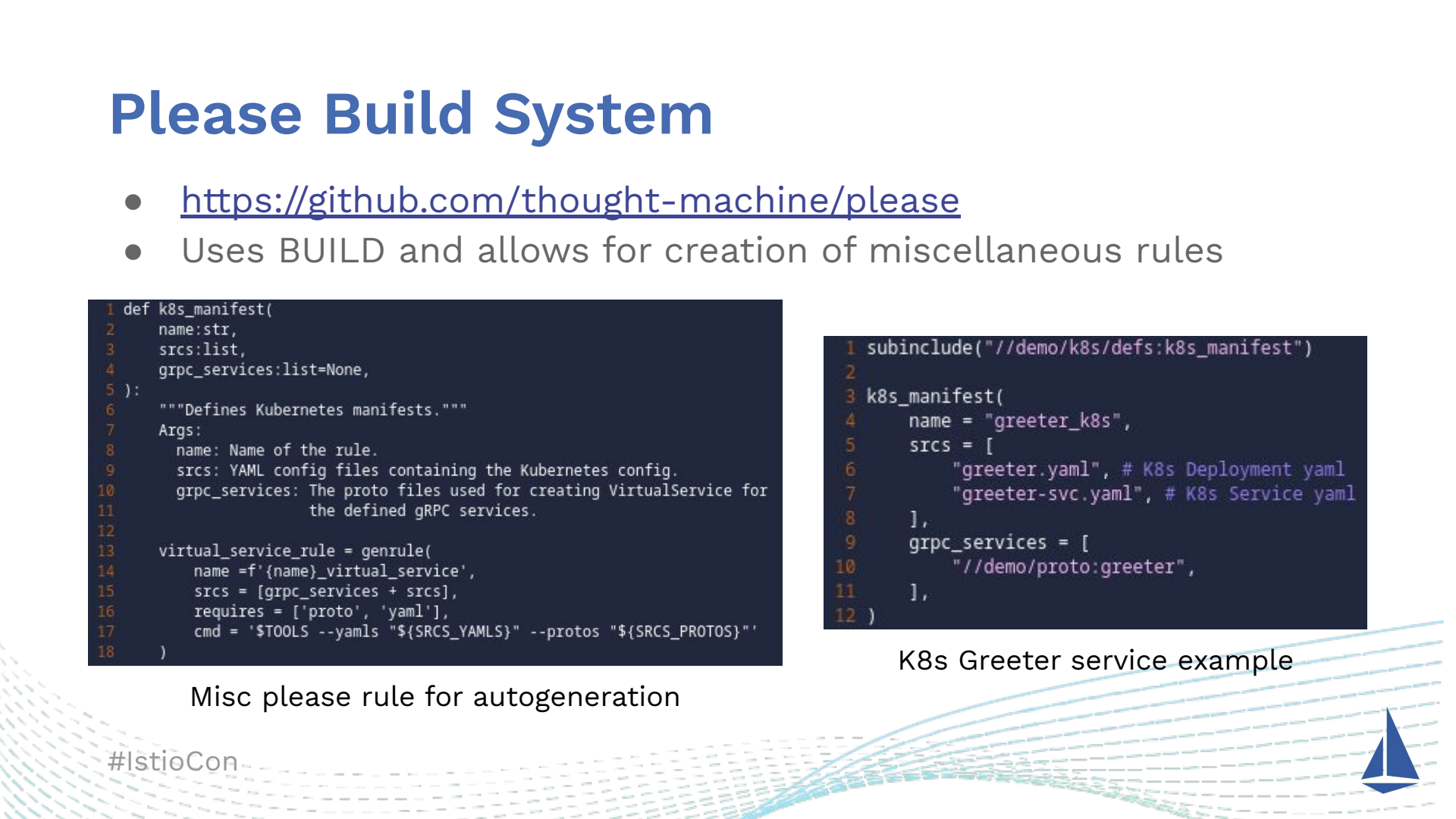
- https://github.com/thought-machine/please
- Uses BUILD and allows for creation of miscellaneous rules

```
def k8s_manifest(
    name:str,
    srcs:list,
    grpc_services:list=None,
):
    """Defines Kubernetes manifests."""
    Args:
      name: Name of the rule.
      srcs: YAML config files containing the Kubernetes config.
      grpc_services: The proto files used for creating VirtualService for
                     the defined gRPC services.

    virtual_service_rule = genrule(
        name =f'{name}_virtual_service',
        srcs = [grpc_services + srcs],
        requires = ['proto', 'yaml'],
        cmd = '$TOOLS --yamls "${SRCS_YAMLS}" --protos "${SRCS_PROTOS}"'
    )
```

Misc please rule for autogeneration

```
subinclude("//demo/k8s/defs:k8s_manifest")

k8s_manifest(
    name = "greeter_k8s",
    srcs = [
        "greeter.yaml", # K8s Deployment yaml
        "greeter-svc.yaml", # K8s Service yaml
    ],
    grpc_services = [
        "//demo/proto:greeter",
    ],
)
```

K8s Greeter service example



Building the new rule

```
➜  src git:(demo) (⎈ microk8s:default) plz build //experimental/vlad/demo/k8s:greeter_k8s_virtu
al_service 2>/dev/null
```


Deploying to a cluster

```
➜  src git:(demo) (⎈ microk8s:default) plz sef deploy-service  //experimental/vlad/demo/k8s:gre
eter_k8s 2> /dev/null
```


- Easy way to manage Virtual Service configs.
- Virtual Service configs become a release artifact.
- Easy abstraction for defining timeouts and retries in a language agnostic way.
- Application developers using Istio/Envoy for retries and timeouts without knowing it.

Thank you!

vlad@thoughtmachine.net

#IstioCon