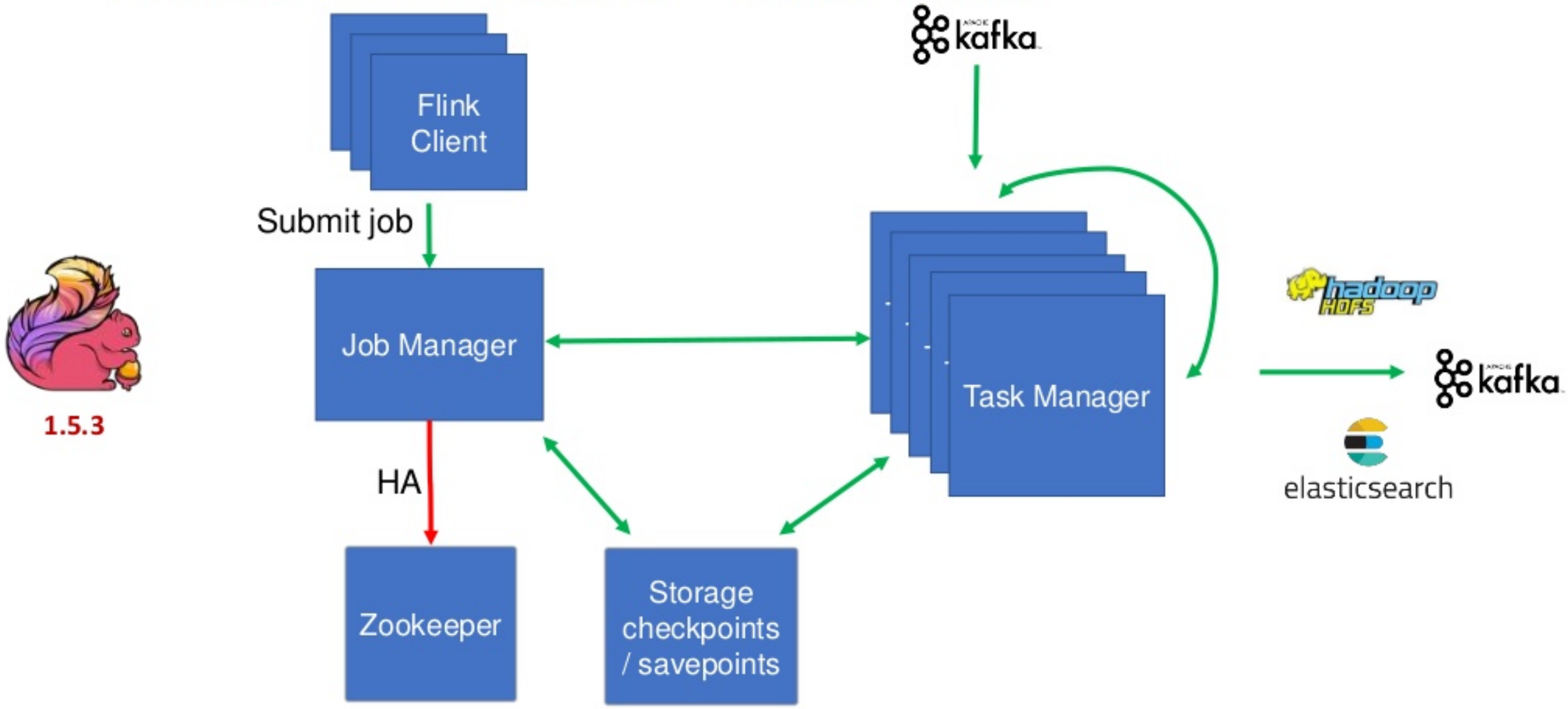
Deploying a secured Flink cluster on Kubernetes

Edward Rojas Clavijo
Software Engineer
IBM France

Use case

- Multiple sources of data => Kafka
- Different storage targets
- Multiple environments
- Sensitive customer data => **Secured**
- Highly available
- Fault tolerant

Flink Architecture – use case



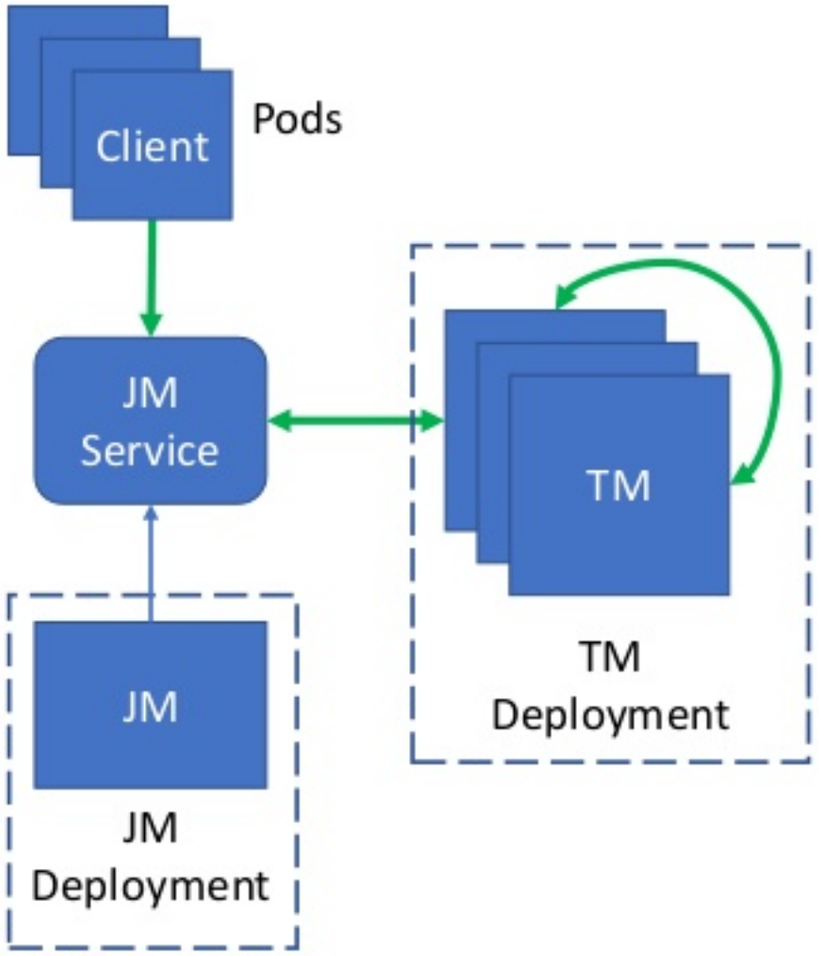
Flink on Kubernetes

- Job Manager Deployment
  - Maintain 1 replica
- Job Manager Service

- Task Manager Deployment
  - Maintain n replicas

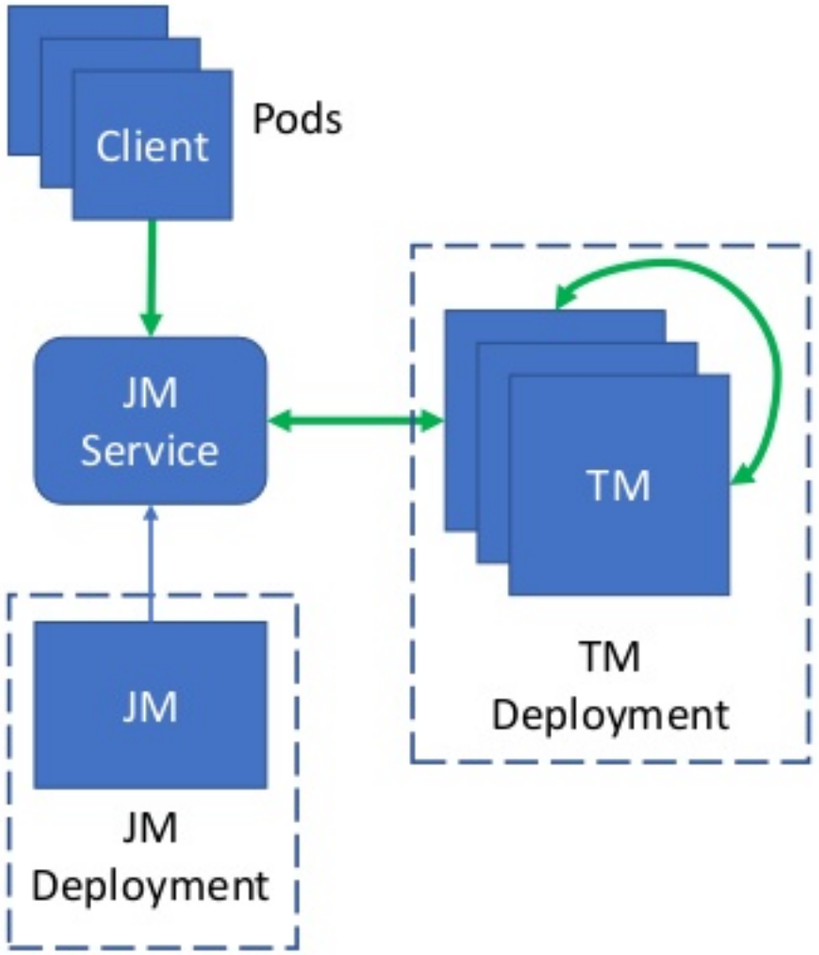
- One Pod per Flink Job
  - Fixed amount of jobs
  - Endless stateful jobs

- ConfigMap for flink conf



Flink SSL/TLS

- Certificates generation
- Dynamic IP
- TM Dynamic hostnames

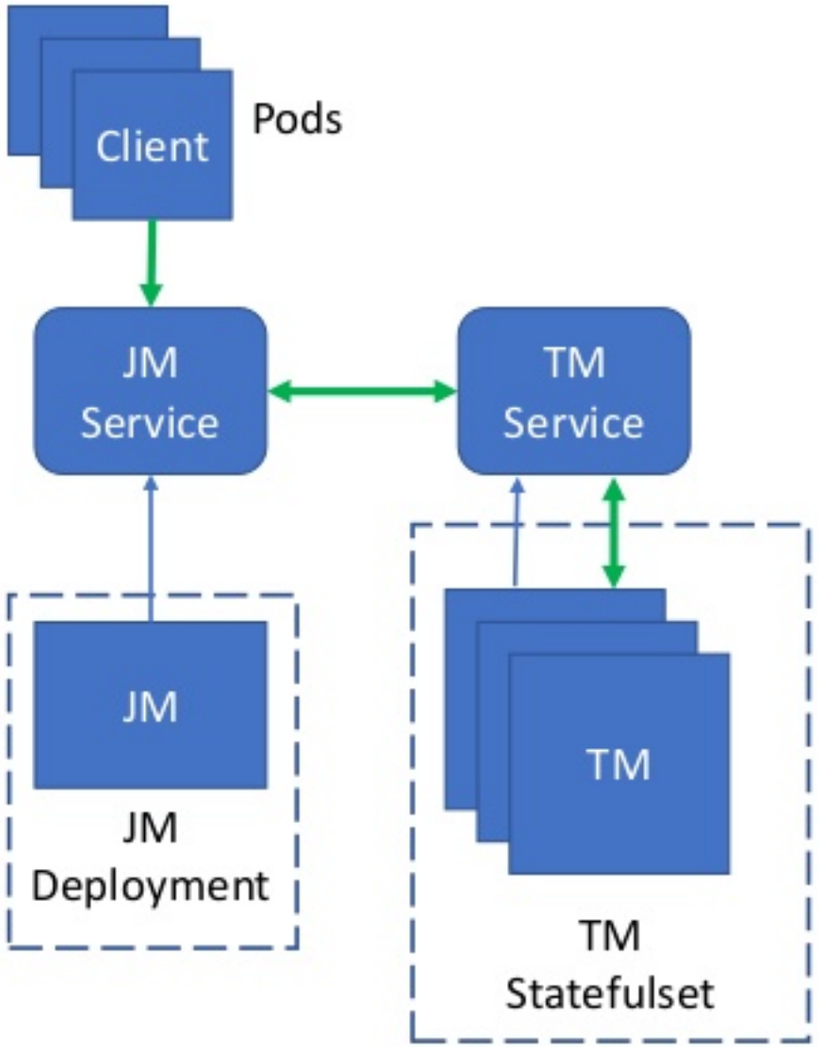


Flink SSL/TLS

- Hostname validation
- Task managers => Statefulset
  - Predictable hostnames
    - **flink-tm-0**.flink-tm-svc.<namespace>.svc.cluster.local
    - **flink-tm-1**.*flink-tm-svc*.<namespace>.svc.cluster.local

TM service

- Wildcard certificate



Flink SSL/TLS

- One single purpose certificate
- Truststore and Keystore with only that certificate
- Deactivate hostname validation

Expose as secrets

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: flink-secrets
type: Opaque
data:
  flink-ssl-password: YmxhYmxhCg==
  flink-ssl-keystore: /u3+7QAAAAIAAA
  flink-ssl-truststore: /u3+7QAAAAIA
```

base64 encoded

Mount on Pods

```yaml
containers:
  ...
  volumeMounts:
  - name: flink-ssl
    mountPath: /etc/flink-ssl
    readOnly: true
...
volumes:
- name: flink-ssl
  secret:
    secretName: flink-secrets
    items:
    - key: flink-ssl-keystore
      path: flink.keystore
    - key: flink-ssl-truststore
      path: flink.truststore
```

Sources and Sinks

Sources and Sinks - Kerberos

- Keytab file exposed as k8s secret
- Configuration properties on ConfigMap => env vars
  - Realm, Principal, KDC
- Dockerfile
  - COPY krb5.conf template – /etc/krb5.conf - set access rights
  - COPY core-site.xml template
- Docker entrypoint:
  - Set krb5.conf
  - Flag for each service
  - Set flink conf properties
  - Set hadoop conf
- Kafka consumer/producer
  - sasl.kerberos.service.name
  - security.protocol => SASL_PLAINTEXT

Sources and Sinks - SSL

- Expose certificates as secrets => mount on pods
- Dockerfile
  - Set access rights to $JAVA_HOME/lib/security/cacerts
- Docker entrypoint:
  - Flag for each service
  - Import certificate into pods trustore

*keytool -importcert -file **${**CERT_FILE} -alias **${**certName} -keystore $JAVA_HOME/lib/security/cacerts -noprompt -storepass changeit*

Sources and Sinks - SSL

- Elasticsearch 6.x => REST
  - ElasticsearchSink – only Flink 1.6

=> Custom ElasticsearchSink

```
builder.setHttpClientConfigCallback(httpClientBuilder ->
    httpClientBuilder
        .setSSLHostnameVerifier(allHostsValid)
        .setDefaultCredentialsProvider(credentialsProvider));
```

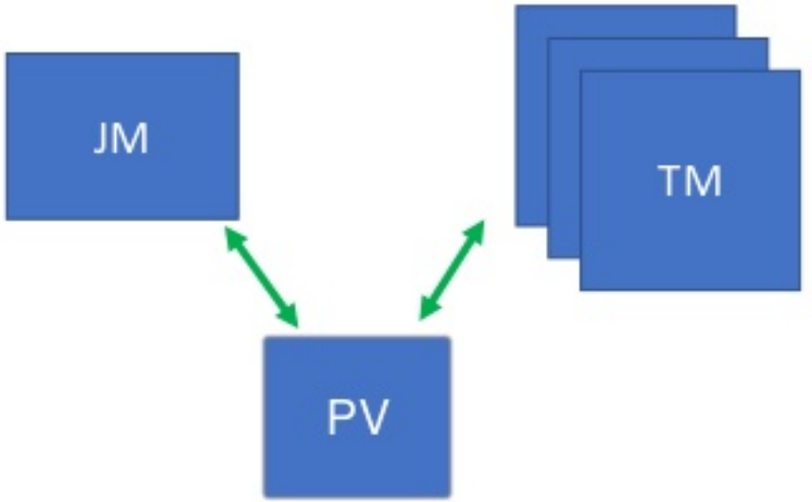
Flink 1.6 =>

```
ElasticsearchSink.Builder<String> esSinkBuilder = new ElasticsearchSink.Builder<>(
    httpHosts, new MyESSinkFunction<String>());

builder.setRestClientFactory(
  restClientBuilder -> {
    restClientBuilder.setHttpClientConfigCallback(...)
  }
);

input.addSink(esSinkBuilder.build());
```

Checkpoints / Savepoints

- Persistent volume
  - NFS
- Encryption in-flight and at-rest
- Encrypt Job state

  => Custom State Serializer



What next?

- Enforce with k8s security config
  - Use RBAC
  - Restrict access to kubectl
  - Use Network Policy
  - Pod security policy
- Single job cluster -> Flink 1.6

Thanks for your attention!

Edward Alexander Rojas Clavijo