ONAP Service Assurance
VES Onboarding, Requirements & Operations

Alok Gupta
+1 (732)-420-7007
ag1367@att.com
Tom Tofigh
+1 (732)-420-7007
mt3682@att.com

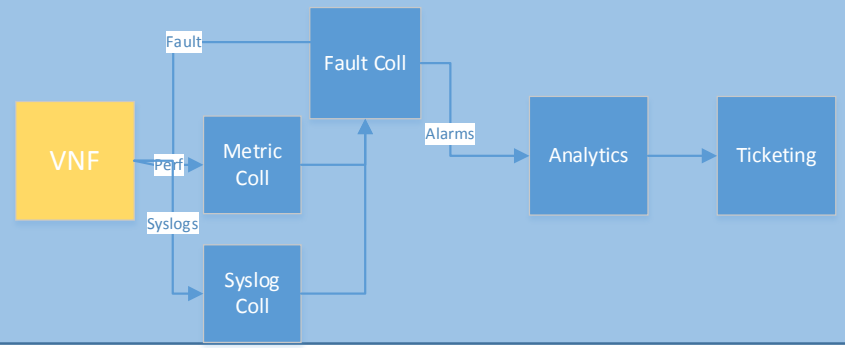
Date , Dec 11th, 2017

Why VES?

Current Environment

Design Time
Interface Specs => Logic in Fault/PM Sub systems

Run Time Environment



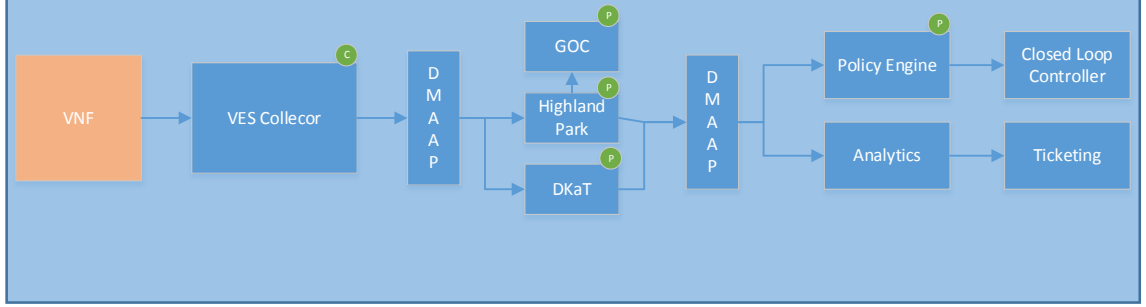
- ❑ Multiple Collectors
- ❑ Need Multiple design-time artifacts
- ❑ Mostly Hard-Coded logic, Some Config.
- ❑ Self-Service Not supported

2 – 3 Months Dev/VNF

VES Environment

Design Time
VES Yaml in SDC => Policy => MS (M-HB, Fault)/CLAMP)

Run Time Environment



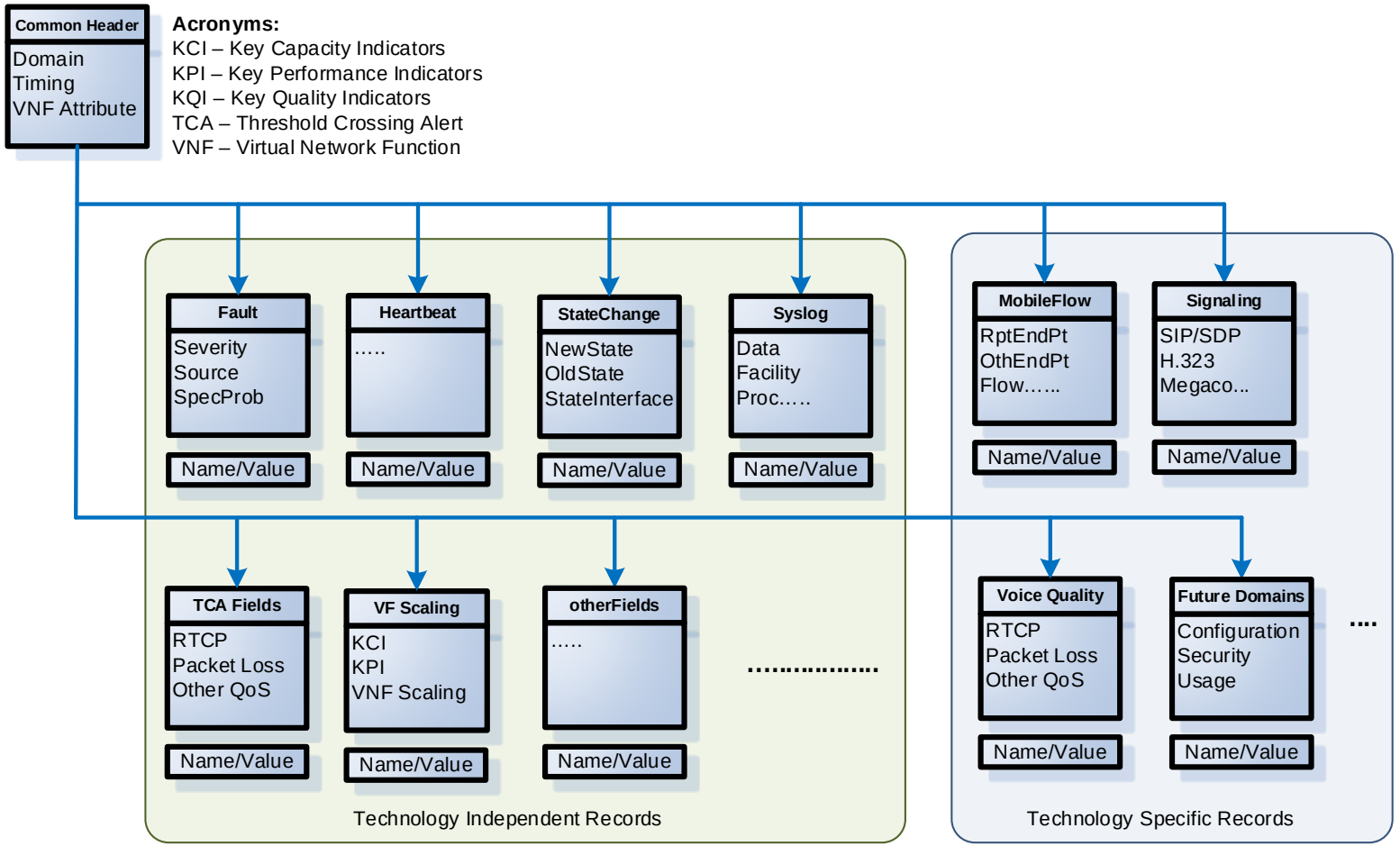
- ❑ One VES collector for all events
- ❑ A YAML Machine Readable Artifact
- ❑ Logic is controlled via policies
- ❑ Supports ECOMP/ONAP Self-Service

Self Serve, Operations Control Via Policy

VES (VF Event Streaming) Progress

- *VES Standard*
  - VES Data Model (1Q2016): The 'Common Event Format' => increased automation in the Management System (DCAE)
  - Standardized VES Internal Header Fields (1Q2016): AT&T-internal data created within DCAE (enrichment, analytics, ticketing, or micro-service) in one data structure
  - VES YAML On-boarding Specification (1Q2017): Machine Readable Event Definition with Actions driving DCAE Flows
- *Open Src Contribution/Influence*
  - VES Agent Libraries – C (2Q2016) and JAVA (2Q2017) https://gerrit.onap.org/r/gitweb?p=demo.git;a=tree;f=vnfs;hb=HEAD
  - Infrastructure Data Collection (CollectD) (1Q2017) - collaboration with Intel
- *VES Telemetry for Tenant and Infrastructure*
  - Holistic Telemetry for Infrastructure: Must include host, hypervisor, vm, switching, data store, resource pool, data center, EPA, Container (K8s), open stack, Contrail.
  - Reuse of DCAE Tools (micro-services, analytics etc.)

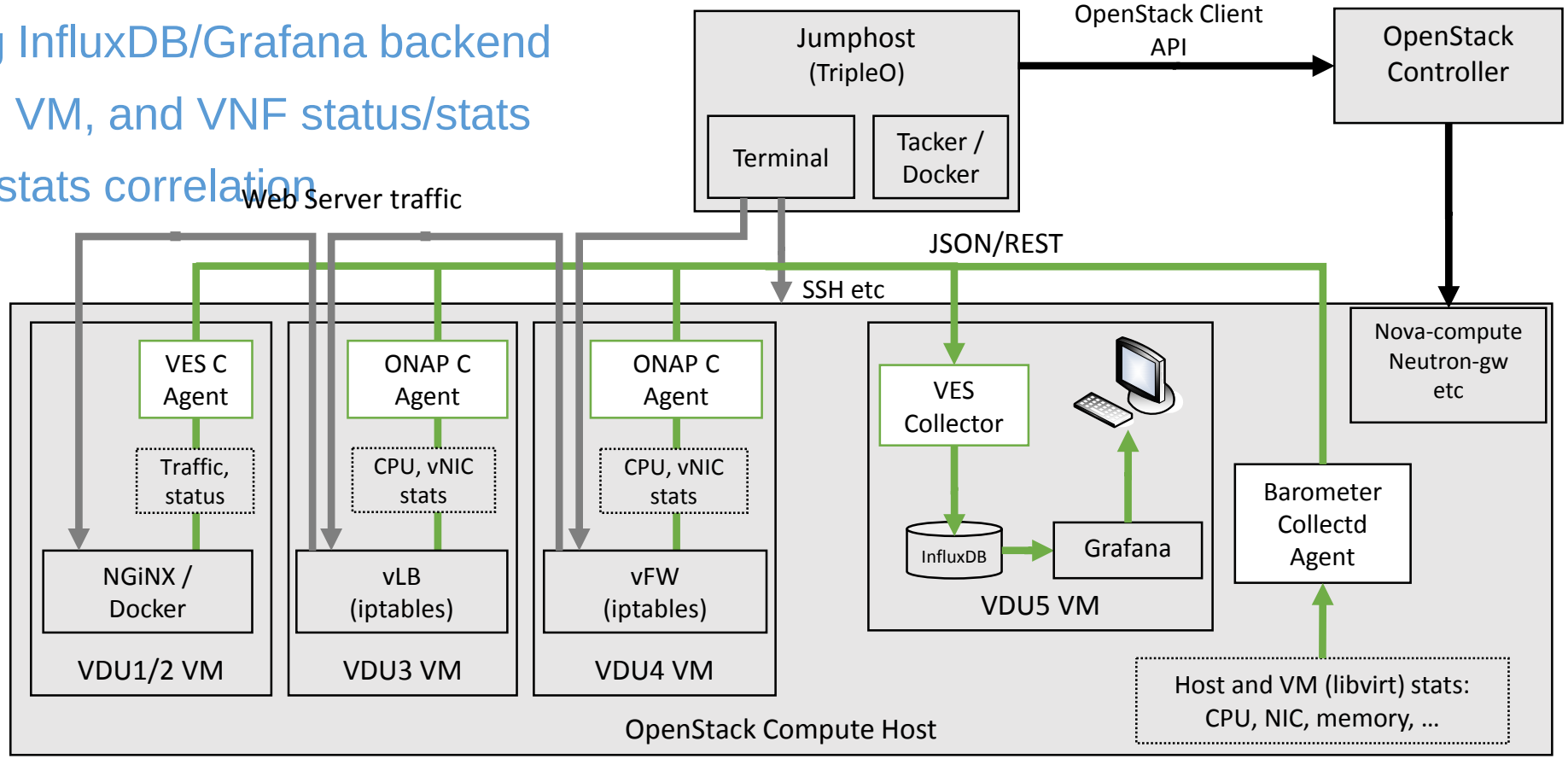
VES – Common Event Data Model



- Common Event Data Model
  - Common Header and Domain Specific Event
  - Extensible for additional fields or domains
- Collector connection and data profile established at VM creation
  - Connection/authentication /profile parameters injected into VM
- Data profile is fully controllable, to optimize telemetry overhead

VES OPNFV Demo

- Leveraging ONAP demo VES Agent and Collector
- Demonstrating InfluxDB/Grafana backend
- Covering host, VM, and VNF status/stats
- Showing fault/stats correlation



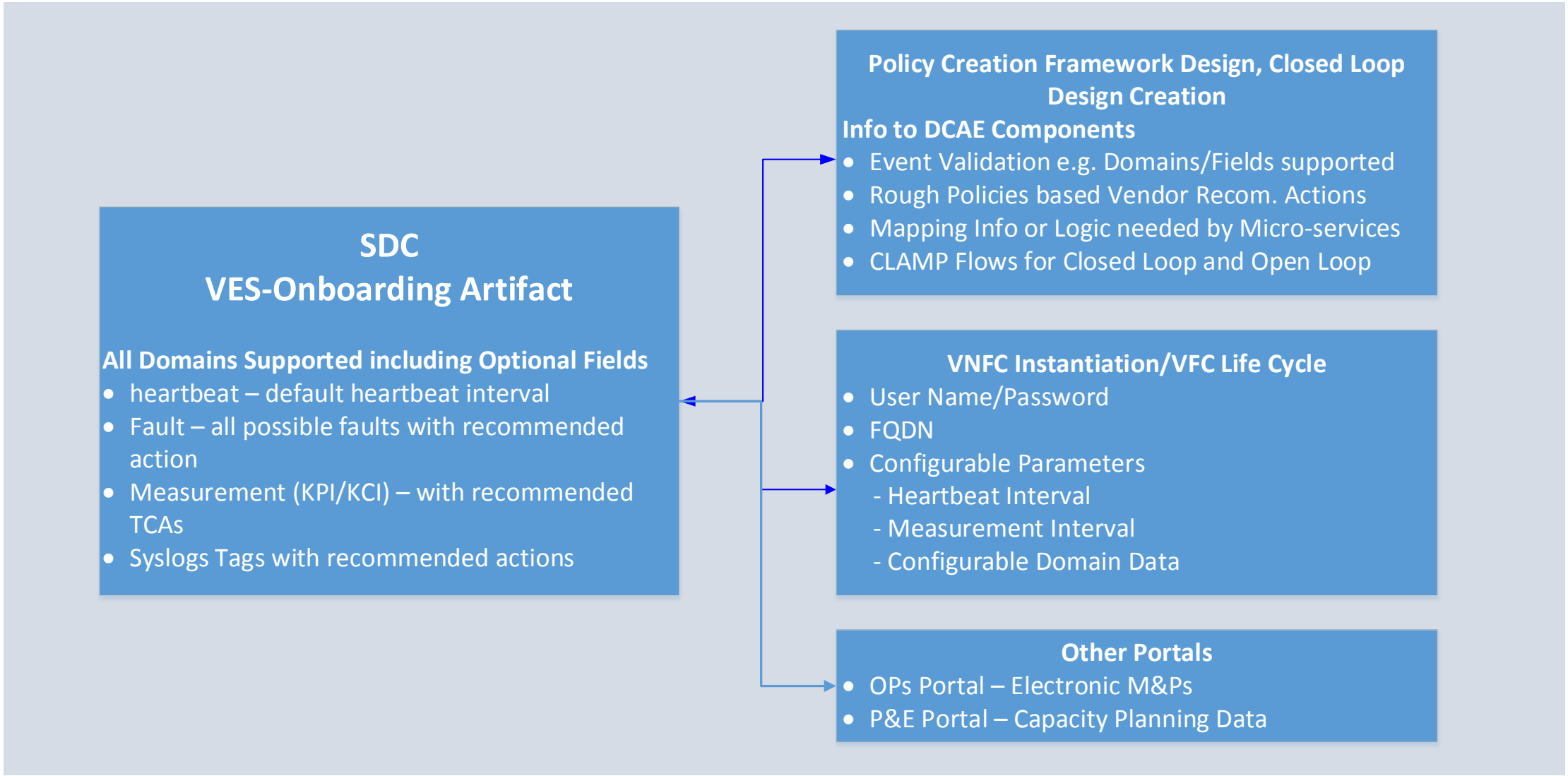
You Tube Demo Video: https://www.youtube.com/watch?v=Zoxcj4mwUwU

VES On-Boarding Artifact (One Document)

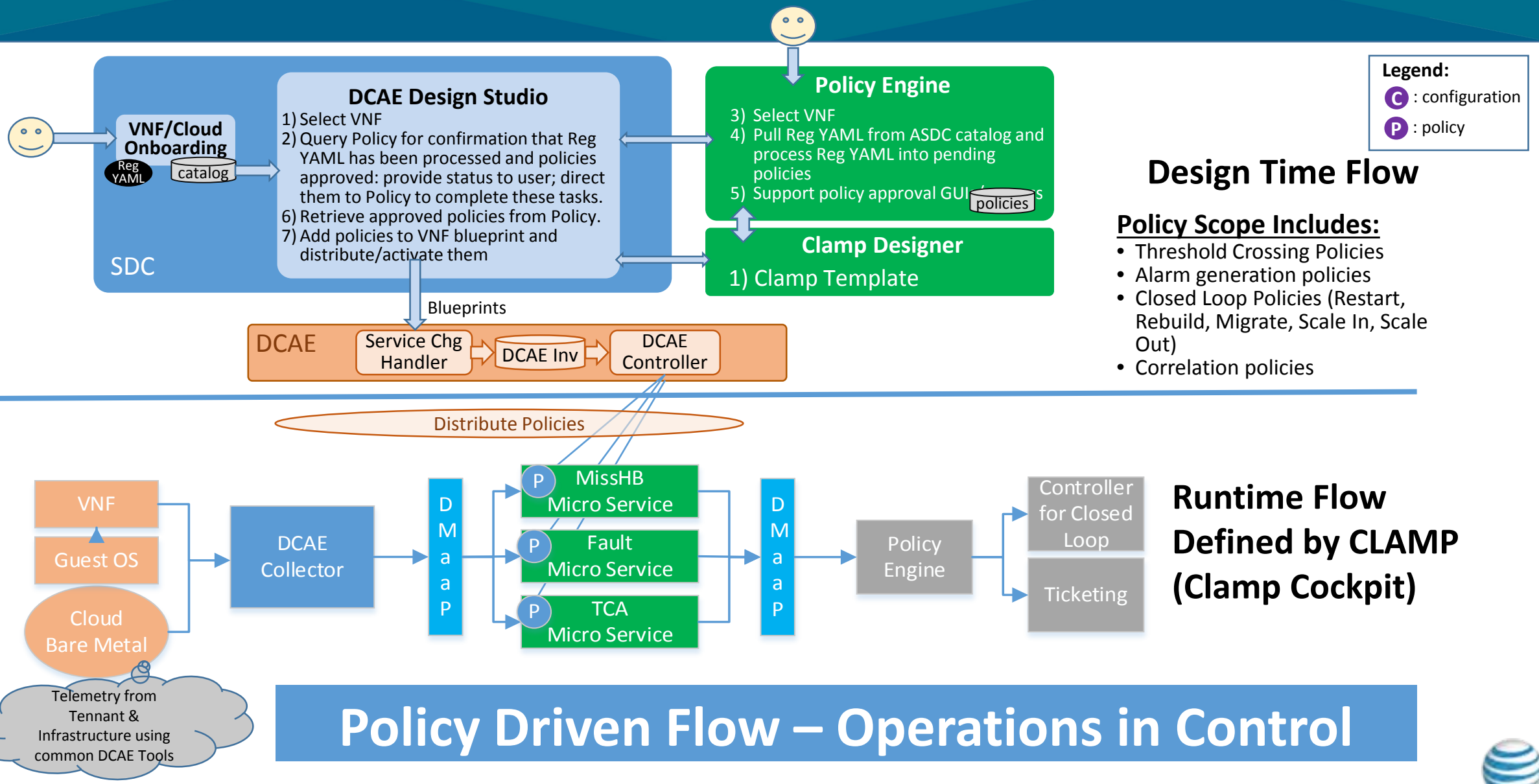
- To enable Self-Service, an on-boarding artifact can be provided by VNF Vendors, covering
  - Which VES event Domains are supported by the VNF's VES agents
  - Optional fields supported, both in the body and as name/value extensions
  - Enumeration of Fault events with recommended action to resolution
  - Ranges and related Thresholding Crossing Alert/Actions for VNF Measurement fields
  - Complex (multi-field correlated) Thresholding Crossing Alert/Actions
  - Recommendation based upon single or correlated fields
  - Syslog Tag data with recommended actions

All Artifacts at one place (SDC), no need for additional documents; Drives Automation

VES On-Boarding Artifact Use



ONAP Self-Serve Event Processing Policy Generation POC



Design Time Flow

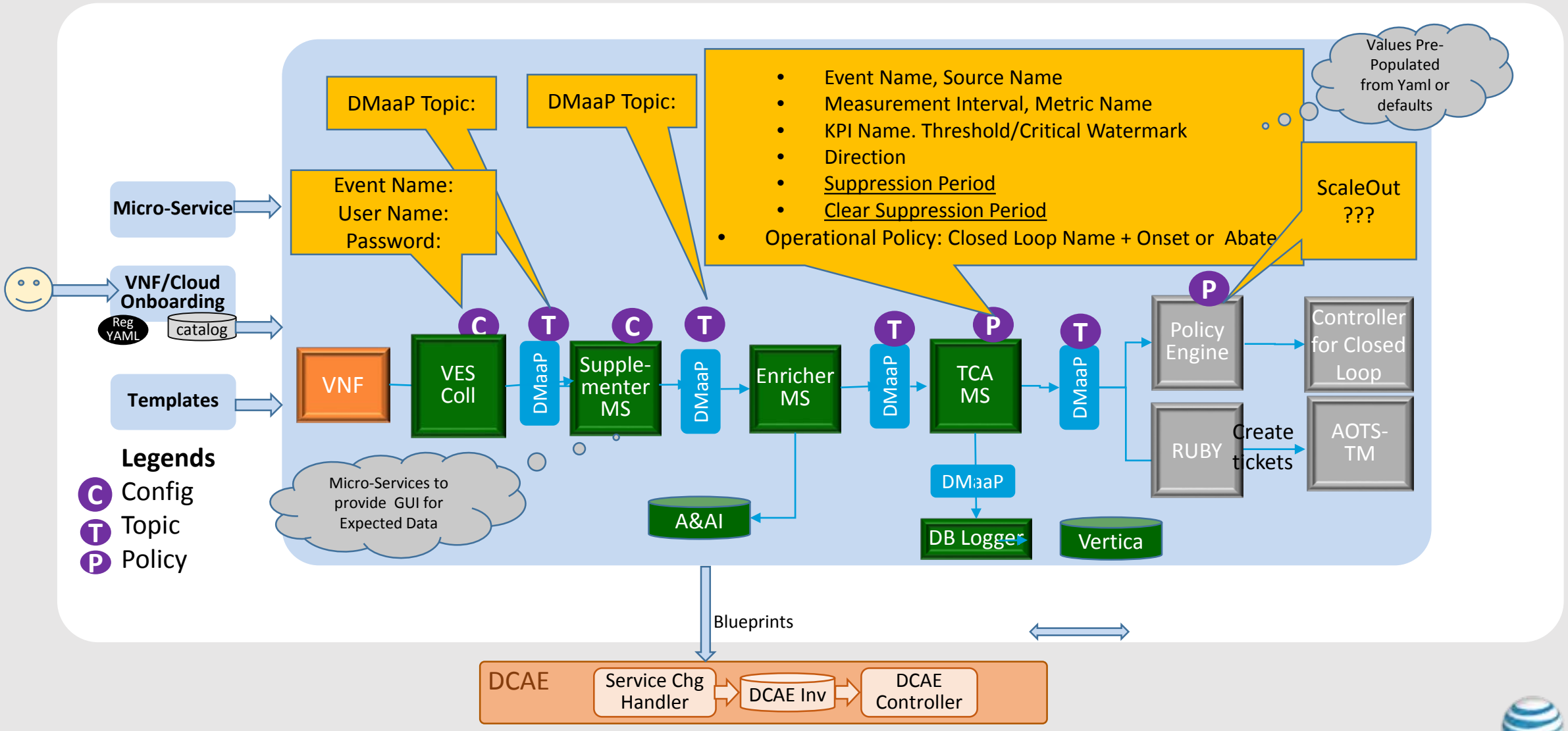
Policy Scope Includes:

- Threshold Crossing Policies
- Alarm generation policies
- Closed Loop Policies (Restart, Rebuild, Migrate, Scale In, Scale Out)
- Correlation policies

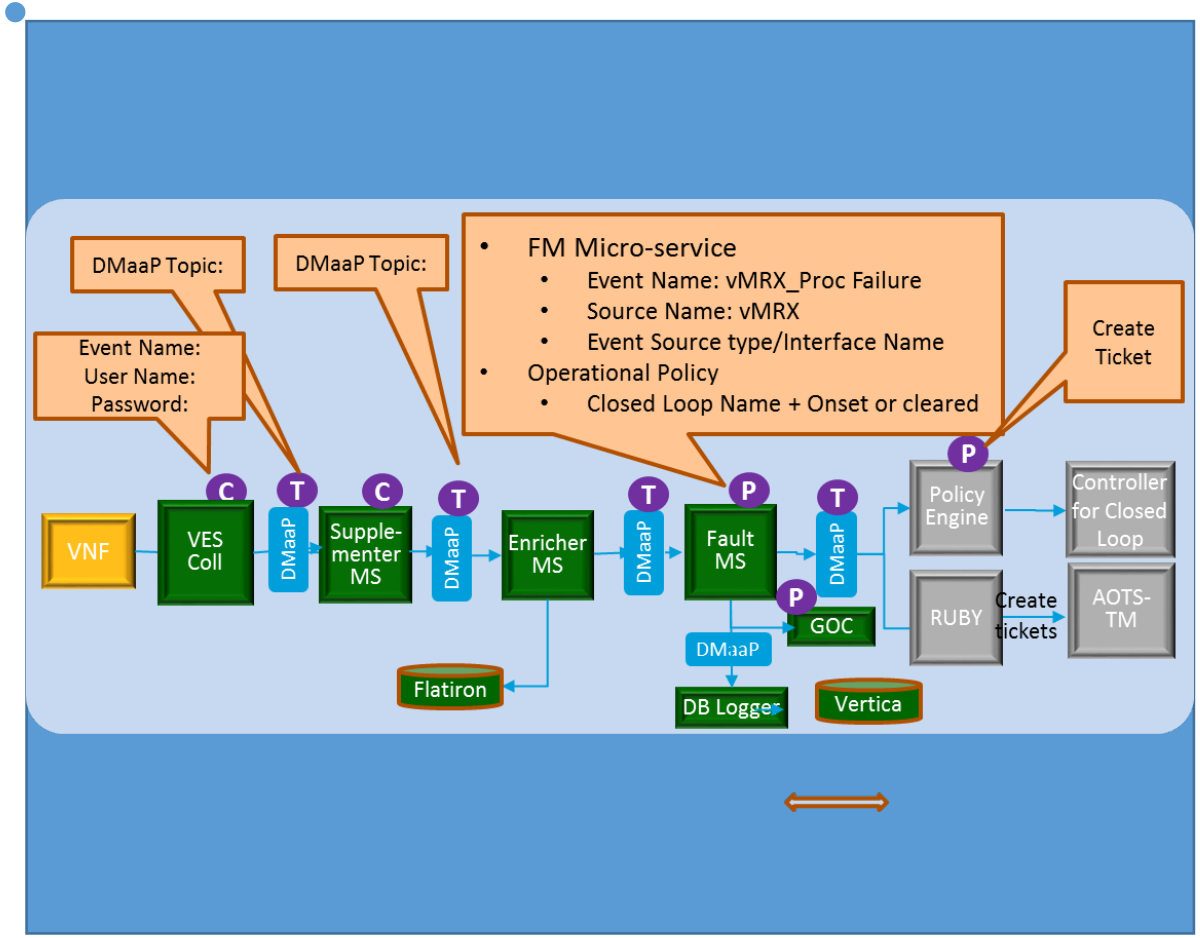
Runtime Flow Defined by CLAMP (Clamp Cockpit)

Policy Driven Flow – Operations in Control

TCA Configuration (One for each TCA being watch and also for complex TCAs)



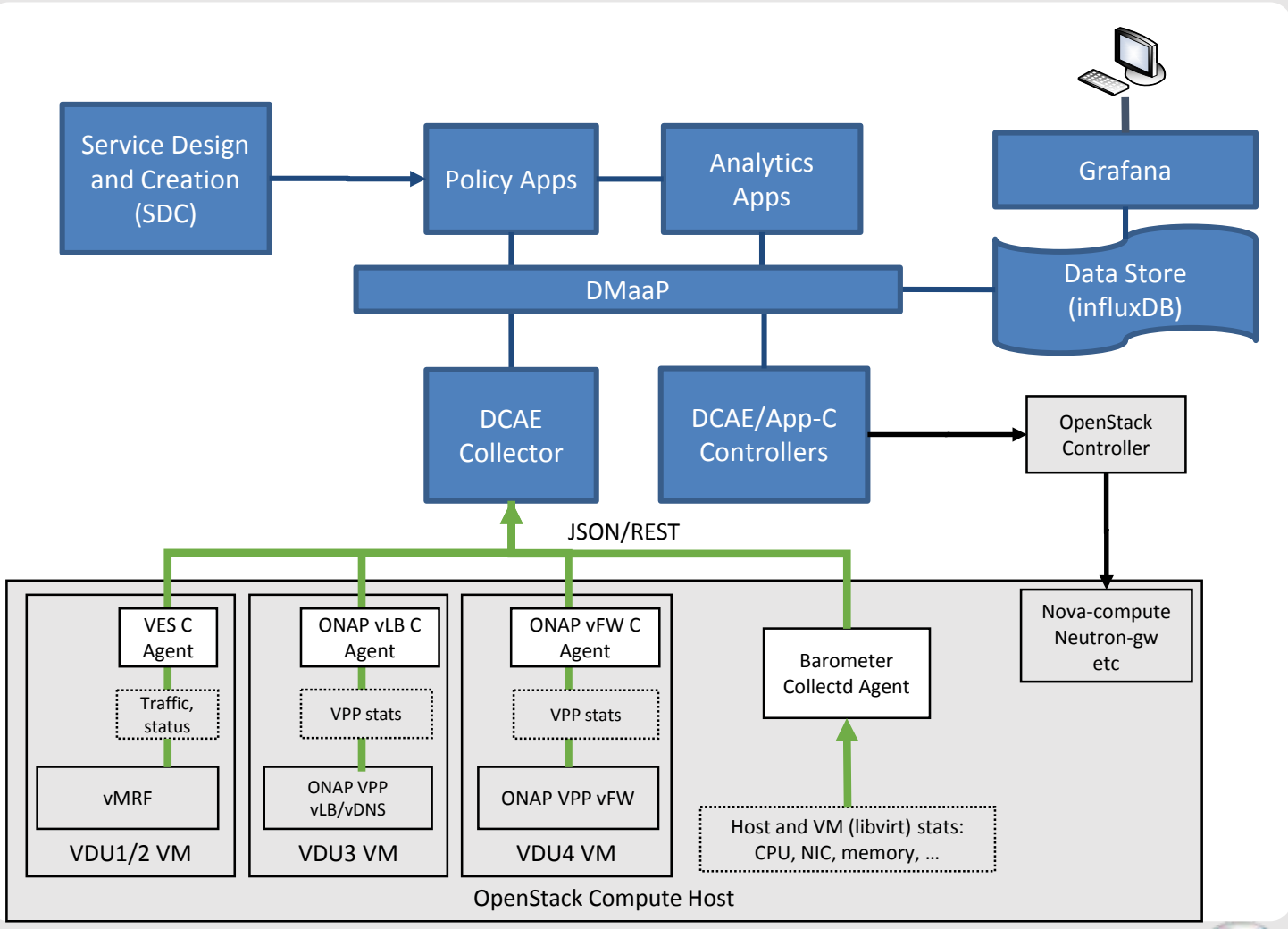
Policy Update GUI




```
event: {presence: required, action: [ any, any, alarm003, RECO-rebuildVnf ],
structure: {
   commonEventHeader: {presence: required, structure: {
      domain: {presence: required, value: fault},
      eventName: {presence: required, value: Fault_vMrf_alarm003},
      eventId: {presence: required},
      priority: {presence: required, value: Medium},
      reportingEntityId: {presence: required},
      reportingEntityName: {presence: required},
      sequence: {presence: required},
      sourceId: {presence: required},
      sourceName: {presence: required},
      startEpochMicrosec: {presence: required},
      lastEpochMicrosec: {presence: required},
      version: {presence: required, value: 3.0}
   }},
   faultFields: {presence: required, structure: {
      alarmCondition: {presence: required, value: alarm003},
      eventSeverity: {presence: required, value: MAJOR},
      eventSourceType: {presence: required, value: virtualNetworkFunction},
      faultFieldsVersion: {presence: required, value: 2.0},
      specificProblem: {presence: required, value: "Configuration file was corrupt
or not present"},
      vfStatus: {presence: required, value: "Requesting Termination"}
    }}
}}
...
---
```

VES SA Proposed E-t-E Architecture

- Create Policies from YAML Artifact
- Policy Hardening (from intent to detailed specification)
- Explore Data Storage
- Build Micro-Services that can be used by Policies
- Build Flows to Analyze Data and Take Action



ONAP Work Items

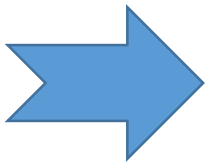
- Enhance the semantics of the Onboarding YAML
- Micro-Services to Provide GUI with Data Input. Need Standardized GUI and API Definitions
- Design Time Flow Creation for a DCAE Flow for every VNF event
- Use Common Vocabulary between Closed Loop with VES Yaml Action
- Simple and Intuitive GUI with pre-populated data
- Same Telemetry for Tenant and Infrastructure
- Next Step - Support policy creation via machine-learning/artificial-intelligence

- Backup Material

Registration – vMRF_heartbeat

Registration Spec

```
---
# registration for Heartbeat_vMRF
event: {presence: required, heartbeatAction: [3, vnfDown, RECO-rebuildVnf],
structure: {
   commonEventHeader: {presence: required, structure: {
      domain: {presence: required, value: heartbeat},
      eventName: {presence: required, value: Heartbeat_vMRF },
      eventId: {presence: required},
      nfNamingCode: {presence: required, value: mrfx},
      priority: {presence: required, value: High},
      reportingEntityName: {presence: required},
      sequence: {presence: required},
      sourceName: {presence: required},
      sourceId: {presence: required},
      startEpochMicrosec: {presence: required},
      lastEpochMicrosec: {presence: required},
      version: {presence: required, value: 3.0}
   }},
   heartbeatFields: {presence: optional, structure:{
      heartbeatFieldsVersion: {presence: required, value: 1.0},
      heartbeatInterval: {presence: required, range: [ 0, 600 ], default:
60 }
   }}
}}
```



Sample Event

```
{
   "event": {
      "commonEventHeader": {
         “domain”: “heartbeat”,
         "eventName": "Heartbeat_vMRF",
         "eventId": "ab305d54-85b4-a31b-7db2-fb6b9e546015",
         “nfNamingCode”: “mrfx”,
         "priority": “High",
         “reportingEntityId”: “cc305d54-75b4-431b-adb2-eb6b9e541234”,
         “reportingEntityName”: “MegaMRFVf”,
         "sequence": 0,
         "sourceId": "de305d54-75b4-431b-adb2-eb6b9e546014",
         “sourceName”: “MegaMRF”,
         “startEpochMicrosec”: 1413378172000000,
         “lastEpochMicrosec”: 1413378172000000,
         “version”: 3.0
      }
   }
}
```

Registering EventType: Fault_vMRF_InvalidLicense

Registration Spec

```
---
# registration for Fault_vMRF_InvalidLicense
event:{ presence: required, action: [any, any, invalidLicense, RECO-renewLicence],
structure: {
   commonEventHeader: {presence: required, structure: {
      domain: {presence: required, value: fault},
      eventName: {presence: required, value: Fault_vMRF_InvalidLicense},
      eventId: {presence: required},
      nfNamingCode: {presence: required, value: mrfx},
      priority: {presence: required, value: High},
      reportingEntityName: {presence: required},
      sequence: {presence: required},
      sourceName: {presence: required},
      startEpochMicrosec: {presence: required},
      lastEpochMicrosec: {presence: required},
      version: {presence: required, value: 3.0}
   }},
   faultFields: {presence: required, structure: {
      faultFieldsVersion: {presence: required, value: 1.2},
      alarmCondition: {presence: required, value: "Invalid license key"},
      eventSourceType: {presence: required, value: virtualNetworkFunction},
      specificProblem: {presence: required, value: "The node license key is invalid"},
      eventSeverity: {presence: required, value: CRITICAL},
      vfStatus: {presence: required, value: Active},
      alarmAdditionalInformation: {presence: required, array: [
         field: {presence: required, structure: {
            name: {presence: required, value: license_key},
            value: {presence: required}
         }}
      ]}
   }}
}}
...
```

Sample Event

```
{
   "event": {
      "commonEventHeader": {
         “domain”: “fault”,
         “eventName”: “Fault_vSCF_InvalidLicense”,
         "eventId": "ab305d54-85b4-a31b-7db2-fb6b9e546015",
         “nfNamingCode”: “mrfx”,
         "priority": "High",
         “reportingEntityId”: “cc305d54-75b4-431b-adb2-eb6b9e541234”,
         “reportingEntityName”: “MegaMRFVf”,
         "sequence": 0,
         "sourceId": "de305d54-75b4-431b-adb2-eb6b9e546014",
         “sourceName”: “MegaMRF”,
         “startEpochMicrosec”: 1413378172000000,
         “lastEpochMicrosec”: 1413378172000000,
         “version”: 3.0
      },
      "faultFields": {
         “faultFieldsVersion”: 1.2
         "alarmCondition": “Invalid license key",
         "eventSourceType": "virtualNetworkFunction",
         "specificProblem": "The node license key is invalid"
         “eventSeverity": "CRITICAL",
         “vfStatus”: “Active”,
         “alarmAdditionalInformation”: [
            {
                  “name”: “license_key”,
                  “value”: “1000”
            }
         ]
      }
   }
}
```

Registering EventType: MFVS vMRF

```
---
# registration for Mfvs_vMRF
event: {presence: required, structure: {
   commonEventHeader: {presence: required, structure: {
      domain: {presence: required, value: measurementsForVfScaling},
      eventName: {presence: required, value: Mfvs_vMRF},
      eventId: {presence: required},
      nfType: {presence: required, value: mrfx},
      priority: {presence: required, value: Normal},
      reportingEntityName: {presence: required},
      sequence: {presence: required},
      sourceName: {presence: required},
      startEpochMicrosec: {presence: required},
      lastEpochMicrosec: {presence: required},
      version: {presence: required, value: 3.0}
   }},
   measurementsForVfScalingFields: {presence: required, structure: {
      measurementsForVfSclaingFieldsVersion: {presence: required, value: 2.0},
      measurementInterval: {presence: required, range: [ 60, 1200 ], default: 180 },
      concurrentSessions: {presence: required},
      cpuUsageArray: {presence: required, array: {
         cpuUsage: {presence: required, structure: {
            cpuIdentifier: {presence: required},
            percentUsage: {presence: required, range: [ 0, 100 ], action: [
                  90, up, CpuUsageHigh, RECO-scaleOut,
                  Tca_vMRF_HighCpuUsage ],
                  action: [25, down, CpuUsageLow, RECO-scaleIn,
                  Tca_vMRF_LowCpuUsage
            ]}
         }}
      }},
      memoryUsageArray: {presence: required, array: {
         memoryUsage: {presence: required, structure: {
            vmIdentifier: {presence: required},
            memoryFree: {presence: required, range: [ 0, 100 ], action: [
                  100, down, FreeMemLow, RECO-scaleOut,
                  Tca_vMRF_LowFreeMemory ], action: [1000, up, FreeMemHigh,
                  RECO-scaleIn, Tca_vMRF_HighFreeMemory
            ]},
            memoryUsed: {presence: required}
         }}
      }},
      numberOfMediaPortsInUse: {presence: required, range: [ 1, 300 ] },
      additionalMeasurements: {presence: required, array: [
         measurementGroup: {presence: required, structure: {
            name: {presence: required, value: licenseUsage},
            measurements: {presence: required, array: [
               field: {presence: required, structure: {
                  name: {presence: required, value: [ G711AudioPort,
                        G729AudioPort, G722AudioPort, AMRAudioPort,
                        AMRWBAudioPort, OpusAudioPort, H263VideoPort,
                        H264NonHCVideoPort, H264HCVideoPort, MPEG4VideoPort,
                        NP8NonHCVideoPort, VP8HCVideoPort, PLC, NR, NG, NLD,
                        G711FaxPort, T38FaxPort, RFactor, T140TextPort ] },
                  value: {presence: required}
               }}
            ]}
         }}
      ]}
   }}
}}
...
```

Registering EventType: Complex TCAs

```
---
# Rules
Rules: [
  rule: {
     trigger: CpuUsageHigh or FreeMemLow,
     microservices: [scaleOut] # Note: this presumes there is a scaleOut microservice
     alerts: [Tca_vMRF_OutOfResources] # Note: this TCA should be defined in the YAML
  },
  rule: {
     trigger: CpuUsageLow && FreeMemHigh,
     microservices: [scaleIn] # Note: this presumes there is a scaleIn microservice
  }
]
...
```

Registering EventType: syslogs vMRF

```
# registration for Syslog_vMRF
# log all, restart if tag = Out_of_Memory
event: {presence: required, action: [any, any, null, RECO-log]
structure: {
   commonEventHeader: {presence: required, structure: {
      domain: {presence: required, value: syslog},
      eventName: {presence: required, value: Syslog_vMRF},
      eventId: {presence: required},
      nfNamingCode: {presence: required, value: mrfx},
      priority: {presence: required, value: Normal},
      reportingEntityName: {presence: required},
      sequence: {presence: required},
      sourceName: {presence: required},
      startEpochMicrosec: {presence: required},
      lastEpochMicrosec: {presence: required},
      version: {presence: required, value: 3.0}
   }},
   syslogFields: {presence: required, structure: {
      eventSourceHost: {presence: required},
      eventSourceType: {presence: required, value: virtualNetworkFunction},
      syslogFacility: {presence: required, range: [0, 23]},
      syslogFieldsVersion: {presence: required, value: 3.0},
      syslogMsg: {presence: required},
      syslogPri: {presence: required, range: [0, 192]},
      syslogProc: {presence: required, range: [0, 65536]},
      syslogSData: {},
      syslogSdId: {},
      syslogSev: {presence: required, range : [0-7]},
      syslogTag: {presence: required, action: ["Out_of_Memory",at,null,reco-restart]},
      syslogVer: {presence: required, value 0}
   }}
}}
…
```

ONAP Vendor Event Listener code on Github and ONAP Gerrit

- Hosted in the ONAP github site:
  https://gerrit.onap.org/r/gitweb?p=demo.git;a=tree;f=vnfs;hb=HEAD
- VES Documentation
  - https://github.com/att/evel-test-collector/tree/master/docs/att_interface_definition
- VES EVEL Demo: https://github.com/att/evel-library/tree/master/code/evel_demo
- VES You tube Video from June 15, 2017 OPNFV, Beijing:
  https://www.youtube.com/watch?v=Zoxcj4mwUwU

Summary

- AT&T Requesting VNF Vendors to provide Fault, Measurement and Syslog as per the definition in the following AID* (includes JSON Schema):


Microsoft Word Document


Microsoft Word Document


Microsoft Word Document


C:\Users\ag1367\ nents\Next Gen SA


Microsoft Excel Worksheet

- AT&T has it's Event Library (Agent and Collector Code) available for VNF Vendors that can be used for integration with their VNF to VES events:
- Following are the Yaml On-boarding files for vFW and vMRF


C:\Users\ag1367\ nents\Next Gen SA


C:\Users\ag1367\ nents\Next Gen SA

- The demo files can be found at: https://gerrit.onap.org/r/gitweb?p=demo.git;a=tree;f=vnfs;hb=HEAD