[Draft] Alpha 2 Analyzer API

** Prior version: [Tentative] Alpha External API for Sizing: https://docs.google.com/document/d/1rIO0yTveW-1WqOO9odkmxAdjRC-Rwog01WcQ_7skdyk/edit?ts=5994afc7#

Application management (P0)

Applications actively managed by Hyperpilot are first-class resources supported by the Analyzer API service. The following are the basic operations for such applications.

Example application.json:

```
{
        "app_id": "tech-demo-xxxxxxxx-xxxx", // generate by the api server
        "name": "tech-demo",    // required field, provided by the user
        "microservices": [
          {
            "name": "goddd",
            "service_id":  "service-xxxxxxxx-xxxx"
          },
          {
            "name": "mongo-serve",
            "service_id":  "service-xxxxxxxx-xxxx"
          },
          {
            "name": "pathfinder",
            "service_id":  "service-xxxxxxxx-xxxx"
          }
        ],
        "type": "long-running",    // required field, can be “long-running” or “batch-processing”
        "slo": {
            "metric": "hyperpilot/goddd/api_booking_service_request_latency_milliseconds",
            "type": "latency",    // can be "latency", "throughput", "execution_time"
            "summary": "quantile_90",
```

```
        "value": 300,
        "unit": "milliseconds"
},
"management_features": [
   {
      "name": "interference_management",   // handles conflicts between multiple
apps
      "status": "Enabled",
      "remediation_policy": [
         {
            "action_name": "move_container",
            “mode": "Semi-Auto",    // can be “Manual”, “Semi-Auto”, “Full-Auto”
             "constraints": {
                "key1": "value1",
                "key2": "value2",
                ……
             }
          },
         {
            "action_name": "resize_node",
            “mode": "Manual",
            "constraints": {...}
            }
         }
      ]
   },
   {
      "name": "bottleneck_management":    // handles resource bottleneck of a single
app/microservice/pod/container/node
      "status": "Enabled",
      "remediation_policy": [...]
   },
   {
      "name": "efficiency_management":    // handles low utilization cases
```

```
            "status": "Disabled",
        }
    ],
    "state":  "Registered"    // state can be “Registered”, “Active”, “Unregistered”
}
```

Create a new application

- URL: /v1/apps
- Method: POST
- URL params: none
- Data params: application.json (see above example)
- Success Response: code 200, data: application.json with unique app_id (generated by the api service)
- Error response: TBD (duplicate app name; mandatory “name” or “type” field missing; other error)
- Notes: Even though we have an app_id, we decided to enforce the uniqueness of the app name as well during new app creation.

Get all applications

- URL: /v1/apps
- Method: GET
- URL params: -
- Data params: -
- Success Response: code 200, data: array of application.json for all existing apps
- Error response: TBD (other error)
- Notes:

Get an application via app_id

- URL: /v1/apps/{app_id}
- Method: GET
- URL params: app_id (unique string, required)
- Data params: -
- Success Response: code 200, data: application.json
- Error response: TBD (app id does not exist; other error)
- Notes:

Get an application via app_name

- URL: /v1/apps/info/{app_name}
- Method: GET
- URL params: app_name (unique string, required)
- Data params: -
- Success Response: code 200, data: application.json
- Error response: TBD (app name does not exist; other error)
- Notes:

Update an application via app_id

- URL: /v1/apps/{app_id}
- Method: PUT
- URL params: app_id (unique string, required)
- Data params: application.json (may be partially populated)
- Success Response: code 200, data: updated application.json
- Error response: TBD (app id does not exist; other error)
- Notes:

Delete an application via app_id

- URL: /v1/apps/{app_id}
- Method: DELETE
- URL params: app_id (unique string, required)
- Data params: -
- Success Response: code 200, data: TBD
- Error response: TBD (app id does not exist; other error)
- Notes:

Application microservices management (P0)

Each application contains a collection of microservices running in one or more containers. The notion of a “microservice” in HyperPilot maps to a Service/Deployment/StatefulSet under certain namespace in a K8s cluster. The HyperPilot UI can let the user choose which of these managed entities belong to a particular application, queries HyperPilot Operator to gather the specs of these managed entities, and then uses the analyzer API to create/update/retrieve them in the k8s_services collection in the configdb. In addition, an array of microservices is added

as a sub_resource to the target application in the configdb, each with a reference to a unique service_id in the k8s_services collection.

Therefore, there are two sets of APIs for managing application services, as follows:

1) API for manipulating the k8s_services collection;
2) API for for manipulating the services array of a particular application

Services API-1): Managing the k8s_services collection:

Here're three examples of a k8s_service.json, one for each type:

```
{
    "service_id": "service-xxxxxxxx-xxxx",
    "namespace": "xxxx",
    "kind": "Service",
    "k8s_spec": {
       "metadata": {
          "name": "mysql-service"
       },
       "spec": {
          "selector": {
             "app": "mysql"
          },
          "ports": [
             {
                "protocol": "TCP",
                "port": 3306,
                "targetPort": 3306
             }
          ]
       },
       "status": {}
    },    // end of k8s_spec
    "service_status": "Running"    //    can be “Running”, “Pending”, etc.
}

{
    "service_id": "service-xxxxxxxx-xxxx",
```

```
    "namespace": "xxxx",
    "kind": "Deployment",
    "k8s_spec": {
        "metadata": {
            "name": "nginx"
        },
        "spec": {
            "replicas": 3,
            "template": {
                "metadata": {
                    "labels": {
                        "service": "http-server"
                    }
                },
                "spec": {
                    "containers": [
                        {
                            "name": "nginx",
                            "image": "nginx:1.10.2",
                            "imagePullPolicy": "IfNotPresent",
                            "ports": [
                                {
                                    "containerPort": 80
                                }
                            ]
                        }
                    ]
                }
            },
        "status": {}
        }
    },        // end of k8s_spec
    // "service_status": "Running"
}

  {
      "service_id": "service-xxxxxxxx-xxxx",
      "namespace": "xxxx",
      "kind": "StatefulSet",
      "k8s_spec": {
          "metadata": {
```

```
            "name": "zk"
        },
        "spec": {
            "selector": {},
            "serviceName": "zk-hs",
            "replicas": 3,
            "updateStrategy": {
                "type": "RollingUpdate"
            },
            "template": {
                "metadata": {},
                "spec": {
                    "affinity": {},
                    "containers": [
                        {
                            "name": "kubernetes-zookeeper",
                            "imagePullPolicy": "Always",
                            "image": "gcr.io/google_containers/kubernetes-zookeeper",
                            "resources": {},
                            "ports": []
                        }
                    ]
                }
            }
        },
    "status": {}
    },    // end of k8s_spec
    // "service_status": "Running"
}
```

Create a new k8s_service

- URL: /v1/k8s_services
- Method: POST
- URL params: none
- Data params: k8s_service.json (see above example)

- Success Response: code 200, data: k8s_service.json with unique service_id (generated by the api service)
- Error response: TBD (mandatory fields missing)
- Note: The additional "service_status" field is being descoped in Alpha 2, so the analyzer will not compute a service_status for each k8s_service, or use it to derive and overall status for the application. This means, relevant k8s_services of all status will be included in Hyperpilot.

Get all k8s_services

- URL: /v1/k8s_services
- Method: GET
- URL params: none
- Data params: -
- Success Response: code 200, data: k8s_services.json
- Error response: TBD
- Notes:

Get a k8s_service via service id

- URL: /v1/k8s_services/{service_id}
- Method: GET
- URL params: service_id (unique string, required)
- Data params: -
- Success Response: code 200, data: k8s_service.json
- Error response: TBD (service id does not exist; other error)
- Notes:

Update a k8s_service via service_id

- URL: /v1/k8s_services/{service_id}
- Method: PUT
- URL params: service_id (unique string, required)
- Data params: k8s_service.json
- Success Response: code 200, data: updated k8s_service.json
- Error response: TBD (service id does not exist; other error)
- Notes:

Get a k8s_service via service id

- URL: /v1/k8s_services/{service_id}
- Method: DELETE

- URL params: service_id (unique string, required)
- Data params: -
- Success Response: code 200, data: TBD
- Error response: TBD (service id does not exist; other error)
- Notes:

Services API-2): Managing the microservices sub-resource of an application

Example of a microservice.json:

```
{
   "name": "goddd",
   "service_id":  "service-xxxxxxxx-xxxx"    // ref. id in the k8s_services collection
},
```

Add microservices to an application

- URL: /v1/apps/{app_id}/microservices
- Method: POST
- URL params: app_id (unique string, required)
- Data params: services.json (array of microservice.json's; see above example)
- Success Response: code 200, data: updated application.json
- Error response: TBD (app id does not exist; duplicate microservices; other error)
- Notes:

Get an application's microservices

- URL: /v1/apps/{app_id}/microservices
- Method: GET
- URL params: app_id (unique string, required)
- Data params: -
- Success Response: code 200, data: microservices.json
- Error response: TBD (app id does not exist; microservices not created; other error)
- Notes:

Get an application's microservice via service name

- URL: /v1/apps/{app_id}/microservices/{service_name}
- Method: GET

- URL params: app_id & service_name
- Data params: -
- Success Response: code 200, data: microservice.json
- Error response: TBD (app id does not exist; service name doesn't exist; other error)
- Notes:

Update an application's microservices

- URL: /v1/apps/{app_id}/microservices
- Method: PUT
- URL params: app_id (unique string, required)
- Data params: microservices.json
- Success Response: code 200, data: updated application.json
- Error response: TBD (app id does not exist; microservices not created; other error)
- Notes: The input json does not need to contain all the current microservices; those microservices with the matching "name" will be updated while leaving the other microservices untouched. [**Q: Do we allow new microservices to be added to an array, or only existing ones can be updated?**]

Application features management (P0)

Each application managed by Hyperpilot can benefit from three management features - interference management, bottleneck management, and efficiency management. The user can enable/disable a specific feature in Hyperpilot or change the remediation mode for each feature. For each remediation option, the mode can be "Disabled", "Manual", "Semi-Auto", and "Full-Auto", and the default mode is set to "Semi-Auto" when an application is registered with HyperPilot and the respective feature is enabled, until the user updates it.

The management feature specifications are a sub-resource for an application and are included in application.json. An example of management_features.json follows.

```
{
   "management_features": [
      {
         "name": "interference_management",
```

```
        "status": "Disabled",
        "remediation_policy": [    // mode can be “Manual”, “Semi-Auto”, “Full-Auto”
           {
              "action_name": "move_container",
                 “mode": "Full-Auto",
                 "constraints": {
                      "key1": "value1",
                      "key2": "value2",
                      ……
                 }
              },
              {
                 "action_name": "resize_container",
                 "mode": "Semi-Auto",
                 "constraints": {
                    "key1": "value1",
                    "key2": "value2",
                    ……
                 }
              },
              {
                 "action_name": "resize_node",
                 “mode": "Manual",
                 "constraints": {
                    "key1": "value1",
                    "key2": "value2",
                    ……
                 }
              }
        ]
},
{
        "name”: “bottleneck_management”,
        "status": "Enabled",
```

```
    "remediation_policy": [
        {
            "action_name": "resize_container",
            “mode": "Semi-Auto",
            "constraints": {}
        },
        {
            "action_name": "resize_node",
            “mode": "Manual",
            "constraints": {}
        },
        {
            "action_name": "scale_service",
            “mode": "Full-Auto",
            "constraints": {}
        },
    ]
},
{
    "name”: “efficiency_management”,
    "status": "Enabled",
    "remediation_policy": [
        {
            "action_name": "resize_container",
            “mode": "Semi-Auto",
            "constraints": {}
        },
        {
            "action_name": "resize_node",
            “mode": "Manual",
            "constraints": {}
        }
    ]
}
```

```
    ]
}
```

Get an application's management features

- URL: /v1/apps/{app_id}/management_features
- Method: GET
- URL params: app_id (unique string, required)
- Data params: -
- Success Response: code 200, data: management_features.json
- Error response: TBD (app id does not exist; other error)
- Notes:

Get info for one of the application's management features

- URL: /v1/apps/{app_id}/management_features/{feature_name}
- Method: GET
- URL params: app_id & feature_name
- Data params: -
- Success Response: code 200, data: management_feature.json
- Error response: TBD (app id does not exist; feature name doesn't exsit; other error)
- Notes: "feature_name" can be "interference_management", "bottleneck_management", or "efficiency_management"

Update one of the application's management features

- URL: /v1/apps/{app_id}/management_features/{feature_name}
- Method: PUT
- URL params: app_id & feature_name
- Data params: management_feature.json
- Success Response: code 200, data: updated application.json
- Error response: TBD (app id does not exist; feature name invalid; other error)
- Notes:

Application state management

Each application managed by Hyperpilot has a state, which becomes Registered when the app is first created by the user (with only “name” and “type” as the required fields). An application’s state only becomes “Active” when the user has completed the whole setup workflow. Hyperpilot will only start monitor, analyze, and control an application when it’s in “Active” state. When a user removes the application from Hyperpilot, the state becomes “Unregistered”.

Note: In Alpha 2, we will not tie an application’s overall state to the running status of any microservices under the application, in the spirit of being inclusive. The definition of an application state may be refined in later versions.

The state is a sub-resource for an application and is included in application.json.

Example state.json:

```
{
   "state":  "Registered”    // can be “Registered”, “Active”, or “Unregistered”
}
```

Get an application’s state

- URL: /v1/apps/{app_id}/state
- Method: GET
- URL params: app_id (unique string, required)
- Data params: -
- Success Response: code 200, data: state.json
- Error response: TBD (app id does not exist; other error)
- Notes:

Update an application’s state

- URL: /v1/apps/{app_id}/state
- Method: PUT
- URL params: app_id (unique string, required)
- Data params: state.json
- Success Response: code 200, data: updated application.json

- Error response: TBD (app id does not exist; invalid state value; other error)
- Notes:

Application SLO management

Each application managed by Hyperpilot has a service level objective, which contains a service level indicator (SLI) metric (name and type) and its objective value. An SLO is a sub-resource for an application and is included in application.json.

Note: The user is allowed to update an application's SLO over its lifetime.

Example slo.json:

```
"slo": {
        "metric": "TpmC",
        "type": "throughput",    // can be “latency”, “throughput”, “execution_time”
        "summary": "quantile_95"
        "value": 8000,
        "unit": "transactions/m"
    }
```

Add new SLO to an application

- URL: /v1/apps/{app_id}/SLO
- Method: POST
- URL params: app_id (unique string, required)
- Data params: slo.json (see example above)
- Success Response: code 200, data: updated application.json
- Error response: TBD (app id does not exist; SLO already exists; other error)
- Notes:

Get an application's SLO

- URL: /v1/apps/{app_id}/SLO
- Method: GET
- URL params: app_id (unique string, required)
- Data params: -
- Success Response: code 200, data: slo.json

- Error response: TBD (app id does not exist; SLO not created; other error)
- Notes:

Update an application's SLO

- URL: /v1/apps/{app_id}/SLO
- Method: PUT
- URL params: app_id (unique string, required)
- Data params: slo.json
- Success Response: code 200, data: updated application.json
- Error response: TBD (app id does not exist; SLO not created; other error)
- Notes: If it's decided that we maintain a history of past SLOs for each application, then during an update operation, the previous SLO before the update will be moved into the SLO_history collection.

Application service level monitoring

Note: This API is tentative; it may be removed later.
Each application is being monitored by a set of snap agents on the nodes, which also retrieves the measured value of the SLI metric from another APM solution (e.g., Prometheus, statsd, datadog, etc.). The monitoring service offers the following APIs.

Create SLO status for an application

- URL: /v1/apps/{app_id}/SLO_status
- Method: POST
- URL params: app_id (unique string, required)
- Data params: status.json (see example above)
- Success Response: code 200, data: status.json
- Error response: TBD (duplicate, app with the same name already exists?)
- Notes:

Get an application's SLO status

- URL: /v1/apps/{app_id}/SLO_status
- Method: GET
- URL params: app_id (unique string, required)
- Data params: -
- Success Response: code 200, data: status.json

- Error response: TBD (app id does not exist; SLO status unavailable; other error)
- Notes:

Update an application's SLO status

- URL: /v1/apps/{app_id}/SLO_status
- Method: PUT
- URL params: app_id (unique string, required)
- Data params: status.json
- Success Response: code 200, data: updated status.json
- Error response: TBD (app id does not exist, SLO status unavailable, other error)
- Notes:

Detection and diagnosis of application SLO violation

Each application is being managed by a collection of node analyzers (one on each node) and an app-level analyzer regarding the health of its service level and the operation of its service components on different nodes. The node-analyzer examines container-level and node-level performance stats collected within a node, computes a set of derived "health" metrics (e.g., based on individual thresholds) and pushes them down the data pipeline. The app-analyzer detects SLO-violation incidents for a managed app and selects the top three most-correlated health metrics, which will be represented as "problems" of various types. For each problem, a recommendation engine will choose up to three remediation actions. The incident, the list of problems and respective remediation actions will be stored as diagnosis_result in the resultdb in MongoDB.

Example incident.json, diagnosis_result.json, problem.json, remediation_option.json: (Tentative: More relevant fields can be added as appropriate.)

```
"incident": {     // example of an incident of its own, without diagnosis result
                  "id": "incident-xxxx-xxxx",
                  "type": "SLO_violation",
                  "labels": {
                      "app_id": "tech-demo-xxxxxxxx-xxxx"
                  },
```

```
                "metric_name":
                "hyperpilot/goddd/api_booking_service_request_latency_milliseconds",
                "timestamp": xxx,
                "observation_window": 60,    // in seconds
                "severity": 60     // 60% of the samples in the observation window are
                violating the SLO; only report an incident when severity is above 50
            }

"diagnosis_result": {
                        "app_id": "tech-demo-xxxxxxxx-xxxx",
                        "incident_id": “incident-xxxx-xxxx”,
                        "top_related_problems": [
                           {
                              "id": "problem-xxxx-xxxx",
                              "remediation_options": [
                                 { // option-1 },
                                 { // option-2 },
                                 { // option-3 }
                              ]
                           },
                           {
                              "id": "problem-xxxx-xxxx",
                              "remediation_options": [ ]
                           },
                           {
                              "id": "problem-xxxx-xxxx",
                              "remediation_options": [ ]
                           }
                        ],
                        "timeout_window": 300 // how many seconds before this particular
                        diagnosis times out
                    }
```

```
"problem":  {
                    "id": “problem-xxxx-xxxx”,
                    "type": "resource_bottleneck",
                    "labels": {
                        "container_id": "goddd-xxxxxxxx-xxxx",
                        "resource_type": "blkio"
                    },
                    "metric_name":
                    "intel/docker/stats/cgroups/blkio_stats/io_serviced_recursive/value",
                    "threshold": {
                        “type”: “UB”,    // upper bound
                        “value”: 1000
                    },
                    "timestamp": xxx
                    "observation_window": 180    // in seconds
                    "severity": 75     // 75% of the samples in the observation window are above
                    the threshold; ; range 50-100
                }

"remediation_option":  {
                                "action": "move_container",
                                "metadata": {
                                    "container_id": "goddd-xxxxxxxx-xxxx",
                                    "source_node": "node-xxxxxxxx-xxxx",
                                    "destination_node": "node-xxxxxxxx-xxxx"
                                }
                            }

"remediation_option":  {
                                "action": "resize_container",
                                "metadata": {
                                    "container_id": "goddd-xxxxxxxx-xxxx",
                                    "resources": {
```

```
                    "requests": {
                        "cpu": "2000m",
                        "memory": "4096Mi"
                    }
                    "limits": {
                        "cpu": "2500m",
                        "memory": "5192Mi"
                    }
                }
            }
        }

"remediation_option": {
            "action": "scale_service",
            "metadata": {
                "service_name": "goddd",
                "replicas": 3
            }
        }

"remediation_option": {
            "action": "resize_node",
            "metadata": {
                "node_id": "node-xxxxxxxx-xxxx",
                "instance_type": "t2.xlarge"
            }
        }
```


Add a new incident

- URL: /v1/incidents
- Method: POST
- URL params: -

- Data params: incident.json (**see example above**)
- Success Response: code 200, data: new unique incident id (generated by our api service)
- Error response: TBD (app id does not exist; other error)
- Notes: Currently the only type of incidents covered is “SLO_violation”, which is detected by the app-analyzer itself. (Other types of incidents to be added later.)

Get an application’s incidents for a given time window

- URL: /v1/incidents
- Method: GET
- URL params: -
- Data params: app_id, start_time, end_time (default is now-5m to now)
- Success Response: code 200, data: incidents.json
- Error response: TBD (app id does not exist; time window invalid; other error)
- Notes:

Get an application’s most recent diagnosis result

- URL: /v1/diagnosis
- Method: GET
- URL params: -
- Data params: app_id
- Success Response: code 200, data: diagnosis_result.json
- Error response: TBD (app id does not exist; the last diagnosis has timed out; other error)
- Notes:

Management of opportunities (P0) and risks (P1)

When the application is not experiencing an SLO violation, the app-analyzer can still identify certain conditions detected from the node analyzer such as “low_utilization” for a node or container, present them as a set of “opportunities” to the user. In addition, the app-analyzer can summarize “high_utilization” or “node_saturation” types of conditions on nodes and containers related to the application, and present them to the user as “risks”. For each opportunity or risk, it then generates one or more remediation actions and presents them to the user.

Example risks and opportunities:

```
"risk":       {
                    "id": “risk-xxxx-xxxx”,
```

```
                "type": "over_utilization",
                "labels": {
                    "node_id": "nodename-xxxxxxxx-xxxx",
                    "resource_type": "memory”
                },
                "metric_name": "intel/procfs/meminfo/mem_free_perc",
                "threshold": {
                    “type”: “LB”,    // lower_bound
                    “value”: 10
                },
                "timestamp": xxx
                "observation_window": 60    // in seconds
                "severity": 90     // 90% of the samples in the observation window are below
                the threshold; ; range 50-100
            }

"opportunity":    {
                    "id": “opportunity-xxxx-xxxx”,
                    "type": "under_utilization",
                    "labels": {
                        "node_id": "nodename-xxxxxxxx-xxxx",
                        "resource_type": "cpu”
                    },
                    "metric_name": "intel/procfs/cpu/utilization_percentage",
                    "threshold": {
                        “type”: “LB”,    // lower_bound
                        “value”: 30
                    },
                    "timestamp": xxx
                    "observation_window": 300    // in seconds
                    "severity": 80     // 80% of the samples in the observation window are
                    below the threshold; ; range 50-100
                }
```

Add a new risk

- URL: /v1/risks
- Method: POST
- URL params: -
- Data params: risk.json (see **example above**)
- Success Response: code 200, data: new unique risk id
- Error response: TBD (input json invalid; other error)
- Notes:

Get all risks matching labels for a given time window

- URL: /v1/risks
- Method: GET
- URL params: -
- Data params: labels[], start_time, end_time (default is now-5m to now)
- Success Response: code 200, data: risks.json
- Error response: TBD (labels do not match; time window invalid; other error)
- Notes:

Add a new opportunity

- URL: /v1/opportunities
- Method: POST
- URL params: -
- Data params: opportunity.json (see **example above**)
- Success Response: code 200, data: new unique opportunity id
- Error response: TBD (input json invalid; other error)
- Notes:

Get all opportunities matching labels for a given time window

- URL: /v1/opportunities
- Method: GET
- URL params: -
- Data params: labels[], start_time, end_time (default is now-5m to now)
- Success Response: code 200, data: opportunities.json
- Error response: TBD (labels do not match; time window invalid; other error)
- Notes: