API Implementation Considerations

Introduction

API *design* guidance can be found on the REST API Guidance page. This page provides guidance on several aspects of API *implementation* at CDC. There are no mandatory components to this guidance, but readers are encouraged to consider the implications of not following this guidance (e.g. development of redundant functionality and difficulty integrating with CDC operations systems).

Version History

Version	Date	Description
0.9	2/8/2019	Initial draft for review by SDP Tech Panel
0.10	2/28/2019	Added implementation language and framework guidance
0.11	5/16/2019	Added additional framework and development tools

Implementation Language and Framework

Language and framework choice will ultimately depend on the preferences and prior experience of team members. However some general guidance applies regardless:

- Use a modern, preferably popular, Web API framework rather than rolling your own. While using a framework may entail additional work to learn how best to use it, the investment will pay off in overall reduced implementation time, ease of onboarding new engineers, and coverage by online communities such as Stack Overflow to help solve issues.
- Ensure good automated integration test coverage of the API using an HTTP client library, don't rely on in-process unit tests. Routing of HTTP requests to the correct handler and deserialization and serialization of requests and responses are common sources of errors, integrations tests will help to catch both kinds of problems.

Some Popular Web API Frameworks

Web API frameworks exist for most popular programming languages and new ones appear regularly. The following is a non-exhaustive list of some popular Web API frameworks by programming language:

- **Ruby**: Sinatra, Rails
- **Python**: Flask, Django
- **Java**: JAX-RS (many implementations of standard API available), Quarkus, Spring, Restlet
- **C#**: ASP.NET Web API, ASP.NET Core MVC
- **Go**: Revel, Gin
- **R**: Plumber
- **JavaScript**: Express, Hapi

Development Tools

In addition to a Web API framework, the following tools may also prove useful:

- Apicurio and SwaggerHub: tools for designing and documenting APIs using the Open API description format
- Postman and SoapUI: tools for API testing
- Microcks: tool for mocking and testing APIs, builds on Postman or SoapUI

Monitoring

The ability to monitor services provides many benefits to both the service owner and clients of the service:

- Rapid identification of service issues and outages allows corrective action to be taken before too many users are impacted.
- Tracking and audit of service usage allows service owners to ensure and prove compliance.
- Identification of performance issues and characterization of the operational context (e.g., number of users, requests per second) that caused them allows service owners to target future IT investments (upgrades, migrations) where they will have the most impact.
- Service users can be provided with a reliable and available service.

Monitoring approaches can be broadly categorized as either passive or active. Each approach is described in the following subsections.

Passive Monitoring

Passive monitoring infrastructure deals with information generated by a service in its normal course of execution. The information used in passive monitoring typically takes the form of logs or event streams. Monitoring infrastructure provides aggregation and visualization functionality to enable users to identify issues and track trends. Brice Figureau's 10 Commandments of Logging describes the following logging best practices in detail.

1. Use standard system logging functionality or frameworks
2. Use log levels appropriately
3. Use categories to classify log messages
4. Write meaningful log messages
5. Log messages in English
6. Include context information in log messages
7. Log messages should be machine parseable
8. Log the right amount
9. Think about the end user of the logs when designing log messages
10. Do not assume that logs are only for troubleshooting

A particular service interaction can spawn a cascade of activity in downstream services. It is often useful to be able to link upstream and downstream events or log entries when investigating issues. This capability is known as distributed tracing and is implemented in several open source projects and products. A distributed tracing working group was chartered by the W3C in July 2018 to "define standards for interoperability between tracing tools" and has participation from several product vendors.

At CDC, ITSO operations use Splunk to aggregate and manage logs from multiple systems. ITSO's Enterprise Container Platform as a Service (ECPaaS) includes a complete log aggregation and exploration capability as does the CA Application Gateway managed by MISO.

For local log management a good starting point is the ELK Stack that is comprised of three open source projects: Elasticsearch, Logstash and Kibana. Often the Logstash component is replaced with Fluentd, a Cloud Native Computing Foundation (CNCF) member project. This variant is refered to as an EFK Stack. ECPaaS uses an EFK Stack for its built in log aggregation and exploration functionality.

Active Monitoring

Active monitoring, often referred to as "health checks," relies on direct interaction between the monitoring infrastructure and a service. It typically takes the form of a monitoring agent invoking functionality provided by the service, ensuring a positive response, and capturing performance metrics.

An IETF internet draft proposes a standard health check response format and a cut-down example is included below for ease of reference.

Example of Health Check Response

```
GET /health HTTP/1.1
Host: example.org
Accept: application/health+json

HTTP/1.1 200 OK
Content-Type: application/health+json
Cache-Control: max-age=3600
Connection: close

{
  "status": "pass",
  "version": "1",
  "releaseID": "1.2.2",
  "notes": [""],
  "output": "",
  "serviceID": "f03e522f-1f44-4062-9b55-9587f91c9c41",
  "description": "health of authz service",
  "details": {
    "cassandra:responseTime": [
      {
        "componentId": "dfd6cf2b-1b6e-4412-a0b8-f6f7797a60d2",
        "componentType": "datastore",
        "observedValue": 250,
        "observedUnit": "ms",
        "status": "pass",
        "time": "2018-01-17T03:36:48Z",
        "output": ""
      }
    ],
    "uptime": [
      {
        "componentType": "system",
        "observedValue": 1209600.245,
        "observedUnit": "s",
        "status": "pass",
        "time": "2018-01-17T03:36:48Z"
      }
    ]
  },
  "links": {
```

```
    "about": "http://api.example.com/about/authz",
    "http://api.x.io/rel/thresholds": "http://api.x.io/about/authz/thresholds"
  }
}
```

Health check agents can be built into the platform that hosts a service (e.g., the application health probes built into ITSO's ECPaaS), or they can be implemented by an external monitoring system such as New Relic or Pingdom. One advantage of the external monitoring approach is that it can detect network issues that may impact external service clients that would be invisible within a hosting platform (e.g., network bottlenecks between external clients and the service).

Privacy Considerations

User privacy needs to be considered when deciding what information to include in logs. IETF RFC 6302 documents logging best practices for internet-facing servers. RFC 6973 offers guidance for incorporating privacy into internet protocols and a new internet draft proposes updates to RFC 6302 based on the guidance from RFC 6973. Readers are encouraged to review the internet draft and consider its recommendations carefully, particularly given the nature of data in use at CDC and new privacy regulations such as the European Union's General Data Protection Rules (GDPR).

Threat Protection

Services hosted on-premise at CDC or in the AHB CDC Cloud Services (ACCS) environment (which includes services hosted on ECPaaS) benefit from the protection provided by the HHS trusted internet connection (TIC). HHS TIC benefits include

- **Packet filtering** - rules-based network access control functionality based on internet protocol addresses and communication sessions
- **Content filtering** - blocking and removal of suspicious content such as spam email and active content that could be used to spread computer viruses
- **Proxying of connections** - hides the details of the CDC network and breaks direct connections between hosts on the internet and within the CDC network
- **Intrusion detection and prevention** - monitoring of events occurring within the CDC network to look for signs of security incidents and providing mechanisms to halt incidents (e.g., blocking suspicious outgoing connections)

Service providers should familiarize themselves with TIC architecture and ensure their systems take full advantage of these capabilities and avoid unnecessarily duplicating those capabilities.

The CA Application Gateway managed by MISO offers additional threat protection capabilities that should be reviewed for applicability.

Rate Limiting / Throttling

Reliability and availability are key desiderata of a service that can be impacted by bad clients that overwhelm a service's capacity. Clients need not be malicious to adversely impact a service. Software bugs or overly enthusiastic adopters can easily generate a surfeit of requests that impact service performance for others. Rate limiting (also called throttling) is a key enabler for reliable and available services.

Rate limiters manage the load on a service and can use different approaches including:

- **Global request rate limiter** - this is the simplest type of limiter, which keeps the number of requests to a server to a specified maximum rate. One disadvantage of this simple approach is that one client can still send many more requests and impact the available request rate for other clients.
- **Per-user/client request rate limiter** - this type of limiter enforces a limit on the number of requests for each client (or user). This prevents one user's requests from impacting others and also allows for tiers of users where some are allowed a higher rate than others (e.g., paid vs free accounts).
- **Concurrent request limiter** - this type of limiter restricts the number of concurrent requests a particular client/user can make. This type of limiter can be particularly useful when an operation takes a long time and some users/clients retry the same request multiple times rather than wait for their original request to complete.
- **Prioritized request limiter** - this type of limiter prioritizes certain types of requests over others. This can be useful to prevent requests from non-essential clients (e.g., reporting) from impacting critical transactional activity.

It may be appropriate to deploy several different types of rate limiters on the same service to deal with all eventualities.

Services deployed in CDC can take advantage of at least two existing rate limiting capabilities:

1. ECPaaS offers both rate and concurrency limiting on API endpoints (see the OpenShift route-specific annotations documentation). ECPaaS also offers connection timeouts that can protect against distributed denial of service (DDoS) attacks.
2. The CA Application Gateway managed by MISO also offers rate and concurrency limiting in addition to throughput quotas.