# OpenWhisk Package Specification

## Version 0.8, Working Draft 07

### *Notational Conventions*

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.



## Introduction

OpenWhisk™ is an open source, distributed Serverless computing project.
Specifically, it is able to execute application logic (*Actions*) in response to events (*Triggers*) from external sources (*Feeds*) governed by simple conditional logic (*Rules*) around the event data.

It provides a programming model for registering and managing *Actions*, *Triggers* and *Rules* supported by a REST-based Command Line Interface (CLI) along with tooling to support packaging and catalog services.

The project includes a catalog of built-in system and utility *Actions* and *Feeds*, along with a robust set of samples that demonstrate how to integrate OpenWhisk with various external service providers (e.g., GitHub, Slack, etc.) along with several platform and run-time Software Development Kits (SDKs).

The code for the Actions, along with any support services implementing *Feeds*, are packaged according to this specification to be compatible with the OpenWhisk catalog and its tooling. It also serves as a means for architects and developers to model OpenWhisk package Actions as part of full, event-driven services and applications providing the necessary information for artifact and data type validation along with package management operations.

## Compatibility

This specification is intended to be compatible with the following specifications:

- *OpenWhisk API which is defined as an OpenAPI document:*
  - https://raw.githubusercontent.com/openwhisk/openwhisk/master/core/controller/src/main/resources/whiskswagger.json
- *OpenAPI Specification when defining REST APIs and parameters:*
  - https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md

# Revision History

| Version | Date | Notes |
|---|---|---|
| 0.8.1 | 2016-11-03 | Initial public point draft, Working Draft 01 |
| 0.8.2 | 2016-12-12 | Working Draft 02, Add. Use cases, examples |
| 0.8.3 | 2017-02-02 | Working Draft 03, Add use cases, examples, $ notation |
| 0.8.4 | 2017-04-18 | Working Draft 04,<br>Support JSON parameter type;<br>Clarify use of Parameter single-line grammar and inferred types.<br>Add support for API Gateway mappings.<br>Add support for Web Actions |
| 0.8.5 | 2017-04-21 | Add support for “dependencies”, that is allow automatic deployment of other OpenWhisk packages (from GitHub) that the current package declares as a dependency. |
| 0.8.6 | 2017-07-25 | Clarified requirements for $ dollar notation.<br>Updated conceptual Manifest/Deployment File processing images. |
| 0.8.7 | 2017-08-24 | Added explicit Application entity and grammar. Added API listing to Package entity. Cleaned up pseudo-grammar which contained various uses of credentials in places not intended. Fixed Polygon Tracking example (indentation incorrect). |

# Table of Contents

*Introduction 1*
Compatibility 1
Revision History 2
*Programming Model 4*
OpenWhisk Entities 4
*Package processing 5*
Conceptual Package creation and publishing 5
Conceptual tooling integration and deployment 6
Conceptual representation 7
Composition 7
Namespacing 7
Entity Names 8
Cardinality 8
A single trigger can be configured to invoke multiple Actions.Definitions 8
*Specification 9*
YAML Types 9
OpenWhisk Types 10
Schema 11
Extended Schema 26
Package Artifacts 30

**Normative References 33**
*Non-normative References 33*

**Scenarios and Use cases 34**
*Usage Scenarios 34*
*Schema Use Cases 35*
"Hello world" 35
Basic Trigger and Rule 36
Github feed 37
Github feed advanced 38
RSS Package 39
Polygon Tracking 40
MQTT Package (tailored for Watson IoT) 43
Check deposit processing with optical character recognition 45

**Event Sources 50**
*Curated Feeds 50*
Alarms 50
Cloudant 51
*Public Sources 51*
GitHub WebHook 51

**Other Considerations 52**
*Tooling interaction 52*
Using package manifest directly from GitHub 52
Using package manifest in archive (e.g., ZIP) file 52
*Simplification of WebHook Integration 52*
*Enablement of Debugging for DevOps 52*

# Programming Model

## OpenWhisk Entities

OpenWhisk uses the following entities to describe its programming model:

### *Action*

A stateless, relatively short-running function invoked as an event handler.
*Note: Typically, an Action's run time is on the order of seconds or even milliseconds, but usually well under the defaulted maximum of 5 minutes.*

### *Trigger*

The name for a class of events. Triggers represent the events (and their data) themselves without any concept of how they were generated.

### *Rule*

A mapping from a Trigger to an Action which may contain simple conditional logic. OpenWhisk evaluates incoming events (that belong to a Trigger) and invokes the assigned Action (event handler).

*Note: Rule conditional logic can be described relative to the event data and the target Action input parameters.*

*Note: In a pub-sub system, a trigger could be viewed as a message topic.*

### *Event Source*

An Event Source is the descriptor (edge) for an Event Producer (or provider). It describes the Event Format(s) produced, as well as any configuration and subscription capablities.

### *Feed*

A Feed is an optional service that allows control operations (e.g., configure, start, stop, pause, resume, etc.) and configuration of Events coming from an Event Source.

*Note: The terms Feed and Trigger are often used interchangeably conversationally (since they both represent a class of events); however, we make this distinction.*

### *Package*

A named, shared collection of Actions and Feeds. The goal of this specification is to describe OpenWhisk packages and their component entities and resources to enable an open-ecosystem.

*Note: Not all actions must belong to packages.*
*Note: Designed as a first-class entity within the OpenWhisk platform to be used by tooling such as catalogs (repositories), associated package managers, installers, etc.*

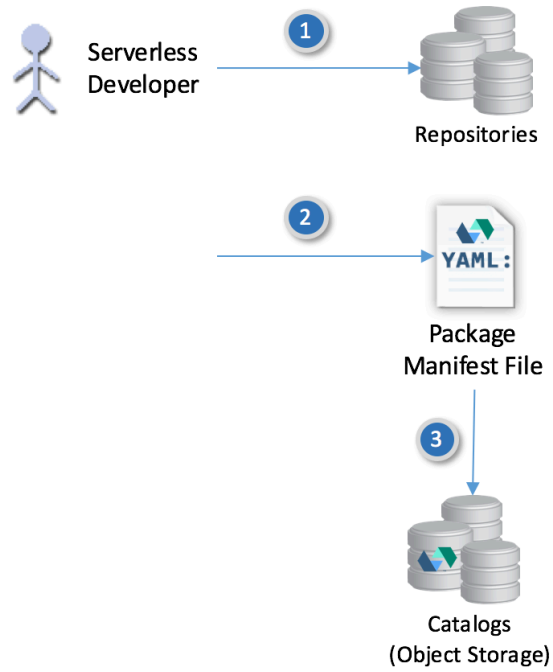
## Package processing

This document defines two file artifacts that are used to deploy Packages to a target OpenWhisk platform; these include:

- **Package Manifest file**: Contains the Package definition along with any included Action, Trigger or Rule definitions that comprise the package. This file includes the schema of input and output data to each entity for validation purposes.
- **Deployment file**: Contains the values and bindings used configure a Package to a target OpenWhisk platform provider's environment and supply input parameter values for Packages, Actions and Triggers. This can include Namespace bindings, security and policy information.

## Conceptual Package creation and publishing

The following diagram illustates how a developer would create OpenWhisk code artifacts and associate a Package Manifest file that describes them for deployment and reuse.

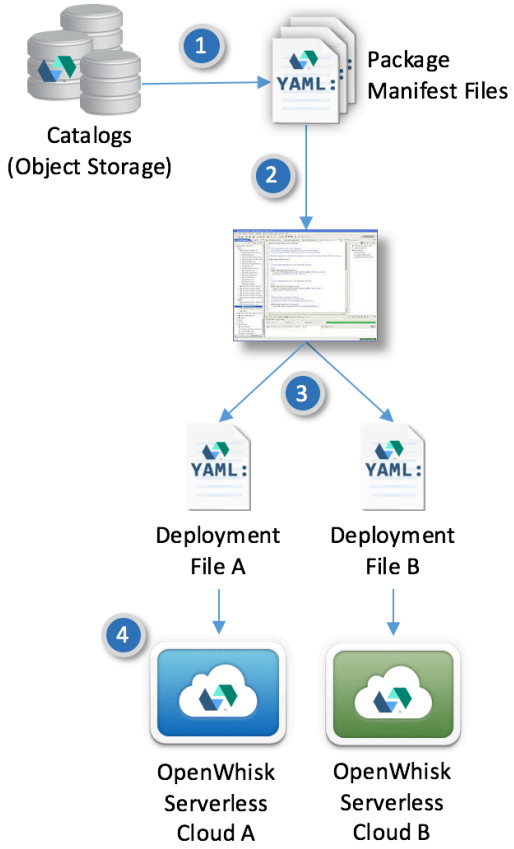



1. Developer ***creates*** and ***reposits*** Serverless code, including:
   - Actions (functions) and
   - Feeds for Event Sources (i.e., Event Providers)
2. Creates a ***Package Manifest File*** which describes the Serverless service's:
   - Repositories (source code locations)
   - Parameter schema (for Actions and Feeds)
   - Configuration and Lifecycle APIs for Feeds
   - Event Sources (and corresponding Event schema)
   - Triggers and Rules
   - Compositions of Actions
   - Annotations (tags, User Interface hints, etc.)
3. Publishes ***Package Manifests*** to enable Catalog features for Serverless services:
   - Automated or manual Discovery and Search
   - Graphical Display / Selection
   - Functional applicability
   - Compositional type validation

## Conceptual tooling integration and deployment

The following diagram illustrates how Package manifests can be leveraged by developer tooling to integrate OpenWhisk Serverless functions.

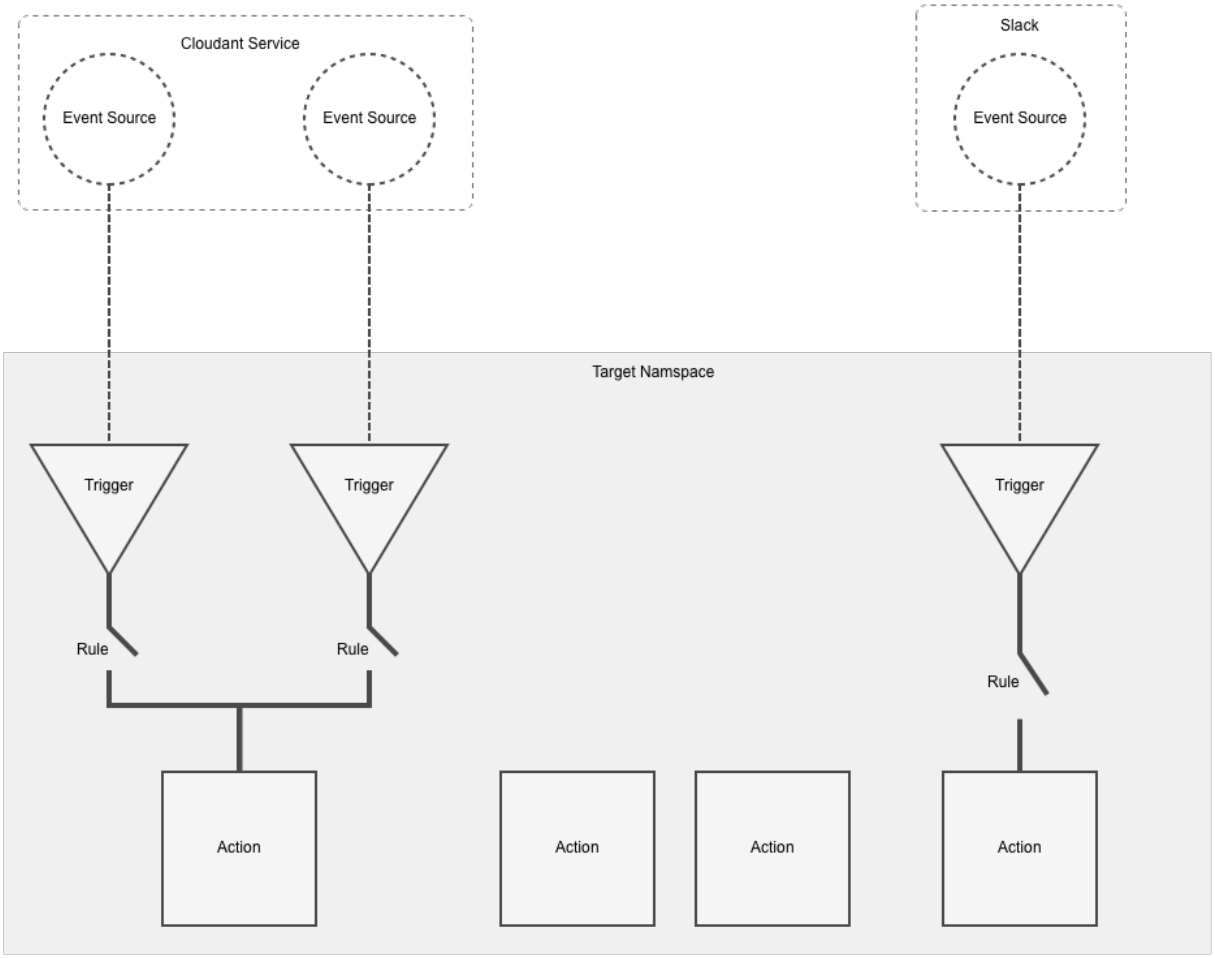
1. Developer ***searches*** and ***discovers*** OpenWhisk packages described by the ***Package Manifest*** in one or more Catalogs, that can:
   - Help analyze, augment and annotate application information and data.
   - Add value added functionality to a base application or workflow.
2. Imports Open ***Package Manifest Files*** and related code and artifacts into development tooling, including:
   - Project and Application (source code) Repositories
   - Integrated Development Environments (IDEs)
   - Cloud-based design, workflow and application workspaces.
3. Creates OpenWhisk ***Deployment Files*** for one or more target OpenWhisk enabled Clouds, with
   - Parameter values for desired target environment
   - Appropriate Credentials and configurations for chosen Event Sources and Feeds.
4. Deploys ***Packages*** (i.e., Actions, Triggers, Feeds, etc.) to OpenWhisk enabled Clouds, using,
   - ***Package Manifest*** and ***Deployment File(s)***.




### *Notes*

- Deployment Files are optional. Deployment can be fully accomplished with simply the Manifest File.

## Conceptual representation




## Composition

### *Action Sequence*

An Action that is a sequenced composition of 2 or more existing Actions. The Action Sequence can be viewed as a named pipe where OpenWhisk can automatically take the output of a first Action 'A' in a declared sequence and provides it as input to the next Action 'B' in the sequence and so on until the sequence completes.

*Note: This composition technique allows the reuse of existing action implementations treating them as "building blocks" for other Actions.*

## Namespacing

Every OpenWhisk entity (i.e., Actions, Feeds, Triggers), including packages, belongs in a *namespace.*

The fully qualified name of any entity has the format:

`/<namespaceName>[/<packageName>]/<entityName>`

The namespace is typically provided at bind-time by the user deploying the package to their chosen OpenWhisk platform provider.

*Note: The `/whisk.system` namespace is reserved for entities that are distributed with the OpenWhisk system.*

## Entity Names

The names of all entities, including actions, triggers, rules, packages, and namespaces, are a sequence of characters that follow the following format:

- The first character SHALL be an alphanumeric character, a digit, or an underscore.
- The subsequent characters MAY be alphanumeric, digits, spaces, or any of the following:

  `_, @, ., -`
- The last character SHALL NOT be a space.

More precisely, a name must match the following regular expression (expressed with Java metacharacter syntax):

```
\A([\w]|[\w][\w@ .-]*[\w@.-]+)\z
```

## Cardinality

### *Trigger to Action*

A single trigger can be configured to invoke multiple Actions.**Definitions**

### *Activation*

An invocation or "run" of an action results in an activation record that is identified by a unique activation ID. The term Activation is short-hand for the creation of this record and its information.

### *Repository*

A location that provides storage for sets of files, as well as the history of changes made to those files.

### *Project*

A description of a software application which enables management of its design, implementation, source control, monitoring and testing.

### *Application*

A computer program designed to perform a group of coordinated functions, tasks, or activities to achieve some result or user benefit.

### *[Cloud] Service*

Any resource, including a functional task, that is provided over the Internet. This includes delivery models such as *Platform as a Service* (PaaS), *Infrastructure as a* Service (IaaS), as well as *Serverless*.

## Specification

This specification utilizes the YAML language, a superset of JSON, which supports key features for packaging descriptors and configuration information such as built-in data types, complex data types, anchors (relational information), files, comments and can embed other data formats such as JSON and XML easily.

### YAML Types

Many of the types we use in this profile are *built-in* types from the YAML 1.2 specification (i.e., those identified by the "tag:yaml.org,2002" version tag).

The following table declares the valid YAML type URIs and aliases that SHALL be used when defining parameters or properties within an OpenWhisk package manifest:

| Type Name | Type URI | Notes |
|---|---|---|
| `string` | tag:yaml.org,2002:str (default) | Default type if no type provided |
| `integer` | tag:yaml.org,2002:int | Signed. Includes large integers (i.e., long type) |
| `float` | tag:yaml.org,2002:float | Signed. Includes large floating point values (i.e., double type) |
| `boolean` | tag:yaml.org,2002:bool | This specification uses lowercase 'true' and lowercase 'false' |
| `timestamp` | tag:yaml.org,2002:timestamp (see YAML-TS-1.1) | ISO 8601 compatible. |
| `null` | tag:yaml.org,2002:null | Different meaning than an empty string, map, list, etc. |
| `version` | tag:maven.apache.org:version (see Maven version) | Typically found in modern tooling (i.e., "package@version" or "package:version" format). |
| `string256` | long length strings (e.g., descriptions) | A string type limited to 256 characters. |
| `string64` | medium length strings (e.g., abstracts, hover text) | A string type limited to 64 characters. |
| `string16` | short length strings (e.g., small form-factor list displays) | A string type limited to 16 characters. |

#### *Requirements*

- The '`string`' type SHALL be the default type when not specified on a parameter or property declaration.
- All '`boolean`' values SHALL be lowercased (i.e., '`true`' or '`false`').

## OpenWhisk Types

In addition to the YAML built-in types, OpenWhisk supports the types listed in the table below. A complete description of each of these types is provided below.

| Type Name | Description | Notes |
|---|---|---|
| `json` | The parameter value represents a JavaScript Object Notation (JSON) data object. | The deploy tool will validate the corresponding parameter value against JSON schema.<br><br>Note: The implied schema for JSON the JSON Schema (see http://json-schema.org/). |
| `scalar-unit` | Convenience type for declaring common scalars that have an associated unit. For example, "10 msec.", "2 Gb", etc.) | See description below. |
| `schema` | The parameter itself is an OpenAPI Specifcation v2.0 **Schema Object** (in YAML formatt) with self-defining schema. | The schema declaration follows the OpenAPI v2.0 specification for Schema Objects (YAML format).<br><br>Specifically, see https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#schemaObject |
| `object` | The parameter itself is an object with the associated defined Parameters (schemas). | Parameters of this type would include a declaration of its constituting Parameter schema. |

### *scalar-unit types*

Scalar-unit types can be used to define scalar values along with a unit from the list of recognized units (a subset of GNU units) provided below.

#### *Grammar*

```
<scalar> <unit>
```

In the above grammar, the pseudo values that appear in angle brackets have the following meaning:

- `scalar`: is a required scalar value (e.g., integer).
- `unit`: is a required unit value. The unit value MUST be type-compatible with the scalar value.

#### *Requirements*

- Whitespace: any number of spaces (including zero or none) SHALL be allowed between the scalar value and the unit value.
- It SHALL be considered an error if either the scalar or unit portion is missing on a property or attribute declaration derived from any scalar-unit type.

#### *Recognized units for sizes (i.e., scalar-unit.size)*

| Unit | Description |
|---|---|
| B | byte |
| kB | kilobyte (1000 bytes) |
| MB | megabyte (1000000 bytes) |
| GB | gigabyte (1000000000 bytes) |
| TB | terabyte (1000000000000 bytes) |

#### *Recognized units for times (i.e., scalar-unit.time)*

| Unit | Description |
|---|---|
| d | days |
| h | hours |
| m | minutes |
| s | seconds |
| ms | milliseconds |
| us | microseconds |

### *Object type example*

The Object type allows for complex objects to be declared as parameters with an optional validateable schema.

```
inputs:
  person:
    type: object
    parameters:
```

## Schema

This section defines all the essential schema used to describe OpenWhisk packages within a manifest.

### *General Requirements*

- All field names in this specification SHALL be case sensitive.

### *map schema*

The Map schema is used to define maps of key values within OpenWhisk entities.

#### *Single-line grammar*

```
{ <key_1>: <value_1>, ..., <key_n>: <value_n> }
```

### *Multi-line grammar*

```
# Where 'key_n' is a type <string> and 'value_n' is type <any>.
<key_1>: <value_1>
...
<key_n>: <value_n>
```

### *Examples*

#### *Single-line*

```
alert_levels: { "high": "red", "medium": "yellow", "low": green }
```

#### *Multi-line*

```
alert_levels:
  "high": "red"
  "medium": "yellow"
  "low": green
```

## *Parameter schema*

The Parameter schema is used to define input and/or output data to be used by OpenWhisk entities for the purposes of validation.

### *Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| type | no | `<any>` | `string` | Optional valid type name or the parameter's value for validation purposes. By default, the type is `string`. |
| description | no | `string256` | N/A | Optional description of the Parameter. |
| value | no | `<any>` | N/A | The optional user supplied value for the parameter.<br>Note: this is not the default value, but an explicit declaration which allows simple usage of the Manifest file without a Deployment file.. |
| required | no | `boolean` | `true` | Optional indicator to declare the parameter as required (i.e., `true`) or optional (i.e., `false`). |
| default | no | `<any>` | N/A | Optional default value for the optional parameters. This value **MUST** be type compatible with the value declared on the parameter's `type` field. |
| status | no | `string` | `supported` | Optional status of the parameter (e.g., `deprecated`, `experimental`). By default a parameter is without a declared status is considered supported. |
| schema | no | `<schema>` | N/A | The optional schema if the '`type`' key has the value '`schema`'. The value would include a **Schema Object** (in YAML) as defined by the OpenAPI Specifcation v2.0. This object is based upon the JSON Schema Specification. |

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| properties | no | `<list of Parameter schema>` | N/A | The optional properties if the '`type`' key has the value '`object`'. Its value is a listing of Parameter schema from this specifcation. |

### *Requirements*

- The "schema" key's value MUST be compatible with the value provided on both the "type" and "value" keys; otherwise, it is considered an error.

### *Notes*

- The "type" key acknowledges some popular schema (e.g., JSON) to use when validating the value of the parameter. In the future additional (schema) types may be added for convenience.

### *Grammar*

#### *Single-line*

```
<parameterName>: <YAML type> | scalar-unit | json
```

- Where `<YAML type>` is inferred to be a YAML type as shown in the YAML Types section above (e.g., string, integer, float, boolean, etc.).
- If you wish the parser to validate against a different schema, then the multi-line grammar MUST be used where the value would be supplied on the keyname "value" and the type (e.g., json) and/or schema (e.g., OpenAPI) can be supplied

#### *Multi-line*

```
<parameterName>:
  type: <any>
  description: <string>
  required: <boolean>
  default: <any>
  status: <string>
  schema: <OpenAPI Schema Object>
```

### *Status values*

| Status Value | Description |
|---|---|
| `supported` (default) | Indicates the parameter is supported. This is the implied default status value for all parameters. |
| `experimental` | Indicates the parameter MAY be removed or changed in future versions. |
| `deprecated` | Indicates the parameter is no longer supported in the current version and MAY be ignored. |

## *Shared schema for all Entities*

The entity schema contains fields that are common to all OpenWhisk entities (e.g., Actions, Triggers, Rules, etc.).

### *Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| version | yes | `version` | N/A | The required version for the Entity. |
| description | no | `string256` | N/A | The optional description for the Entity. |
| displayName | no | `string16` | N/A | This is the optional name that will be displayed on small form-factor devices. |
| annotations | no | TBD | N/A | The optional annotations for the Entity. |
| inputs | no | `list` of `parameter` | N/A | The optional ordered list inputs to the Entity. |
| outputs | no | `list` of `parameter` | N/A | The optional outputs from the Entity. |

### *Grammar*

```
version: <version>
description: <string>
annotations: <map of <string>>
inputs: <list of <parameter>>
outputs: <list of <parameter>>
```

### *Requirements*

- Version values for all entities SHALL be provided.
- Non-required fields MAY be stored as “annotations” within the OpenWhisk framework after they have been used for processing.
- Description string values SHALL be limited to 256 characters.
- DisplayName string values SHALL be limited to 16 characters.
- The lists of inputs and outputs MAY also be described as ordered lists (using the YAML ‘-‘ dash convention).
- Annotations MAY be ignored by target consumers of the Manifest file as they are considered data non-essential to the deployment of management of OpenWhisk entities themselves.
    - Target consumers MAY preserve (persist) these values, but are not required to.
- For any OpenWhisk Entity, the maximum size of all Annotations SHALL be (TBD) characters.

### *Notes*

- Input and output parameters are implemented as JSON Objects within the OpenWhisk framework.
- Several, non-normative Annotation keynames and allowed values for (principally for User Interface (UI) design) may be defined below for optional usage.

## *Action entity*

The Action entity schema contains the necessary information to deploy an OpenWhisk function and define its deployment configurations, inputs and outputs.

### Fields

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| function | yes | `string` | N/A | Required source location (path inclusive) of the Action code either<br>• Relative to the Package manifest file.<br>• Relative to the specified Repository. |
| runtime | no | `string` | N/A | The required runtime name (and optional version) that the Action code requires for an execution environment.<br><br>*Note: May be optional if tooling allowed to make assumptions about file extensions.* |
| limits | no | `map` of <limit keys and values> | N/A | Optional map of limit keys and their values.<br><br>*See section "Valid limit keys" below for a listing of recognized keys and values.* |
| feed | no | `boolean` | `false` | Optional indicator that the Action supports the required parameters (and operations) to be run as a Feed Action. |
| web-export | no | `boolean` | `false` | Optionally, turns the Action into a "*web actions*" causing it to return HTTP content without use of an API Gateway. |

### Requirements

- The Action name (i.e., `<actionName>` MUST be less than or equal to 256 characters.
- The Action entity schema includes all general Entity Schema fields in addition to any fields declared above.
- Supplying a `runtime` name without a version indicates that OpenWhisk SHOULD use the most current version.
- Supplying a runtime *major version* without a *minor version* (et al.) indicates OpenWhisk SHOULD use the most current *minor version*.
- Unrecognized `limit` keys (and their values) SHALL be ignored.
- Invalid values for known `limit` keys SHALL result in an error.
- If the Feed is a Feed Action (i.e., the `feed` key's value is set to `true`), it MUST support the following parameters:
  - **lifecycleEvent**: one of 'CREATE', 'DELETE', 'PAUSE', or 'UNPAUSE'
    - These operation names MAY be supplied in lowercase (i.e., 'create', 'delete', 'pause', etc.).
  - **triggerName**: the fully-qualified name of the trigger which contains events produced from this feed.
  - **authKey**: the Basic auth. credentials of the OpenWhisk user who owns the trigger just mentioned.

### Notes

- The maximum code size for an Action currently must be less than 48 MB.
- The maximum payload size for an Action (i.e., POST content length or size) currently must be less than 1 MB.
- The maximum parameter size for an Action currently must be less than 1 MB.

### *Valid Runtime names*

The following runtime values are currently supported by the OpenWhisk platform.

Each of these runtimes also include additional built-in packages (or libraries) that have been determined be useful for Actions surveyed and tested by the OpenWhisk platform. These packages may vary by OpenWhisk release; details on runtimes and their package conventions and default package (or libraries) here:

| Runtime name | Supported versions | Default | Description |
|---|---|---|---|
| `nodejs` | nodejs@6, nodejs@5 | nodejs@6 | Node.js runtime. |
| `java` | java@8 | java@8 | Java language runtime. |
| `python` | python@2.7 | python@2.7 | Python language runtime. |
| `swift` | swift@3, swift@2 | swift@3 | Swift language runtime. |
| `language:default` | N/A | N/A | Permit the OpenWhisk platform to select the correct default language runtime. |

*Please note, if no value for runtime is supplied, the value 'language:default' will be assumed.*

### *Recognized File extensions*

Although it is best practice to provide a runtime value when declaring an Action, it is not required. In those cases, that a runtime is not provided, the package tooling will attempt to derive the correct runtime based upon the the file extension for the Action's function (source code file). The following file extensions are recognized and will be run on the latest version of corresponding Runtime listed below:

| File extension | Runtime used | Description |
|---|---|---|
| `.js` | nodejs | Node.js runtime. |
| `.java` | java | Java language runtime. |
| `.py` | python | Python language runtime. |
| `.swift` | swift | Swift language runtime. |

### *Valid Limit keys*

| Limit Keyname | Allowed values | Default value | Valid Range | Description |
|---|---|---|---|---|
| `timeout` | scalar-unit.time | 60000 ms | [100 ms, 300000 ms] | The per-invocation Action timeout. Default unit is assumed to be milliseconds (ms). |
| `memorySize` | scalar-unit.size | 256 MB | [128 MB, 512 MB] | The per-Action memory. Default unit is assumed to be in megabytes (MB). |
| `logSize` | scalar-unit.size | 10 MB | [0 MB, 10 MB] | The action log size. Default unit is assumed to be in megabytes (MB). |

| Limit Keyname | Allowed values | Default value | Valid Range | Description |
|---|---|---|---|---|
| `concurrentActivations` | integer | 1000 | *See description* | The maximum number of concurrent Action activations allowed (per-namespace).<br><br>*Note: This value is not changeable via APIs at this time.* |
| `userInvocationRate` | integer | 5000 | *See description* | The maximum number of Action invocations allowed per user, per minute.<br><br>*Note: This value is not changeable via APIs at this time.* |
| `codeSize` | scalar-unit.size | 48 MB | *See description* | The maximum size of the Action code.<br><br>*Note: This value is not changeable via APIs at this time.* |
| `parameterSize` | scalar-unit.size | 1 MB | *See description* | The maximum size<br><br>*Note: This value is not changeable via APIs at this time.* |

### *Notes*

The default values and ranges for limit configurations reflect the defaults for the OpenWhisk platform (open source code). These values may be changed over time to reflect the open source community consensus.

### *Web Actions*

OpenWhisk can turn any Action into a "web action" causing it to return HTTP content without use of an API Gateway. Simply supply a supported "type" extension to indicate which content type is to be returned and indentified in the HTTP header (e.g., `.json`, `.html`, `.text` or `.http`).

Return values from the Action's function are used to construct the HTTP response. The following response parameters are supported in the response object.

1. `headers`: a JSON object where the keys are header-names and the values are string values for those headers (default is no headers).
2. `code`: a valid HTTP status code (default is 200 OK).
3. `body`: a string which is either plain text or a base64 encoded string (for binary data).

### *Grammar*

```
# Note: the optional [.<type>] grammar is used for naming Web Actions.
<actionName>[.<type>]:
  <Entity schema> # Common to all OpenWhisk Entities
  function: <string>
  runtime: <name>[@<[range of ]version>]
  limits:
    <list of limit key-values>
```

```
  feed: <boolean> # default: false
```

### *Example*

```
my_awesome_action:
  version: 1.0
  description: An awesome action written for node.js
  function: src/js/action.js
  runtime: nodejs@>0.12<6.0
  inputs:
    not_awesome_input_value:
      description: Some input string that is boring
      type: string
  outputs:
    awesome_output_value:
      description: Impressive output string
      type: string
  limits:
    memorySize: 512 kB
    logSize: 5 MB
```

## *Trigger entity*

The Trigger entity schema contains the necessary information to describe the stream of events that it represents. For more information see: "Creating Triggers and Rules".

### *Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| feed | no | `string` | N/A | The optional name of the Feed associated with the Trigger. |
| events | no | `list` of `Event` | N/A | The optional list of valid Event schema the trigger supports. OpenWhisk would validate incoming Event data for conformance against any Event schema declared under this key.<br><br>Note: This feature is not supported at this time. This is viewed as a possible feature that may be implemented along with configurable options for handling of invalid events. |

### *Requirements*

- The Trigger name (i.e., `<triggerName>` MUST be less than or equal to 256 characters.
- The Trigger entity schema includes all general Entity Schema fields in addition to any fields declared above.

### *Notes*

- The Trigger entity within the OpenWhisk programming model is considered outside the scope of the Package (although there are discussions about changing this in the future). This means that Trigger and API information will not be returned when using the OpenWhisk Package API:
  - `wsk package list <package name>`
- However, it may be obtained using the Trigger API:

- `wsk trigger list -v`

### *Grammar*

```
<triggerName>:
  <Entity schema> # Common to all OpenWhisk Entities
  feed: my_feed
```

## *Rule entity*

The Rule entity schema contains the information necessary to associates one trigger with one action, with every firing of the trigger causing the corresponding action to be invoked with the trigger event as input. For more information see: "Creating Triggers and Rules".

### *Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| trigger | yes | `string` | N/A | Required name of the Trigger the Rule applies to. |
| action | yes | `string` | N/A | Required name of the Action the Rule applies to. |
| rule | no | `regex` | true | The optional regular expression that determines if the Action is fired.<br><br>Note: In this version of the specification, only the expression "true" is currently supported. |

### *Requirements*

- The Rule name (i.e., `<ruleName>`) MUST be less than or equal to 256 characters.
- The Rule entity schema includes all general Entity Schema fields in addition to any fields declared above.

### *Grammar*

```
<ruleName>:
  <Entity schema> # Common to all OpenWhisk Entities
  trigger: <string>
  action: <string>
  rule: <regex>  # "true" supported at this time
```

### *Example*

```
my_rule:
  desription: Enable events for my Action
  trigger: my_trigger
  action: my_action
```

## *Composition entity*

The Composition entity schema contains information to declare compositions of OpenWhisk Actions. Currently, this includes Action Sequences where Actions can be composed of two or more existing Actions.

### *Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| type | no | `string` | `sequence` | The optional type of Action composition. *Note: currently only 'sequence' is supported.* |
| inputs | no | `list` of `parameter` | N/A | The optional list of parameters for the Action composition (e.g., Action Sequence). |
| sequence | no | `ordered list` of `Action` (names) | N/A | The optional expression that describes the connections between the Actions that comprise the Action sequence composition. |
| parameterMappings | no | TBD | N/A | The optional expression that describes the mappings of parameter (names and values) betweens the outputs of one Action to the inputs of another Action.<br><br>Note: Currently, mappings are not supported and JSON objects are passed between each Action in a sequence. At this time, it is assumed that the Actions in a sequence are designed to work together with no output to input mappings being performed by the OpenWhisk platform. |

### *Requirements*

- The Composition name (i.e., `<compositionName>` MUST be less than or equal to 256 characters.
- The Composition entity schema includes all general Entity Schema fields in addition to any fields declared above.

### *Grammar*

```
<compositionName>:
  <Entity schema> # Common to all OpenWhisk Entities
  type: <string>
  inputs: <list of parameters>
  sequence:
    - <order list of action names>
  parameterMappings:
    # TBD. This is a future use case.
```

### *Example*

```
my_action_sequence:
  type: sequence
  sequence:
    - action_1
    - action_2
```

```
    - action_3
  inputs:
    simple_input_string: string
  outputs:
    annotated_output_string: string
```

### *Feed entity*

The OpenWhisk Feed entity schema contains the information necessary to reference a configurable service that works with an existing network accessible service to produce events on its behalf thereby acting as an Event Source.

These are standalone services unto themselves which the OpenWhisk platform does not currently deploy and run from an OpenWhisk Package. At this time, the Package Manifest simply provides the information to access, configure and manage (via lifecycle operations) the Feed service that is already running elsewhere.

#### *Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| location | no | string | `N/A` | The URL for the Feed service which can be used by the OpenWhisk platform or registration and conifguration. |
| credential | no | string | `N/A` | Contains either:<br>• A credential string.<br>• The optional name of a credential (e.g., token) that must be used to acceess the Feed service. Note: this would be defined elsewhere, perhaps as an input parameter to the Package. |
| operations | no | list of operations | `N/A` | The list of operations (i.e., APIs) the Feed supports on the URL provided described, by default, using the OpenAPI (f.k.a. “Swagger”) specification schema. |
| operation_type | no | openwhisk \| openapi@<version> | `openwhisk` | The specification format for the operation definitions. |
| action | no | string | `N/A` | The optional name of the Action if this is a Feed Action, that is, the Feed service implementaion is an OpenWhisk Action. |

#### *Requirements*

- The Feed name (i.e., `<feedName>` MUST be less than or equal to 256 characters.
- The Feed entity schema includes all general Entity Schema fields in addition to any fields declared above.
- If the `action` field is set, the corresponding Action defintion and function (code) MUST be a valid Feed Action.
- The location and credential SHOULD be supplied if the Feed is not a Feed action from the Deployment File.
- Operation names in manifests MAY be lower or upper cased (e.g., “create” or “CREATE”).

### Grammar

```
<feedName>:
  <Entity schema> # Common to all OpenWhisk Entities
  location: <string>
  credential: <Credential>
  operations:
    <list of operations>
  action: <string>
```

### Example

The following example shows the mandatory operations for Feed Actions.

```
my_feed:
  description: A simple event feed
  location: https://my.company.com/services/eventHub
  credential: my_credential
  operations:
    # Note: operation names in manifests MAY be lower or upper cased.
    create | CREATE:
      inputs:
        <parameters>
    delete | DELETE:
      inputs:
        <parameters>
    pause | PAUSE:
      inputs:
        <parameters>
    unpause | UNPAUSE:
      inputs:
        <parameters>
    # Additional, optional operations
    ...
```

### Discussion

For a description of types of Feeds and why they exist, please see https://github.com/openwhisk/openwhisk/blob/master/docs/feeds.md.

## Feed Actions

OpenWhisk supports an open API, where any user can expose an event producer service as a **feed** in a **package**. This section describes architectural and implementation options for providing your own feed.

### Feed actions and Lifecycle Operations

The *feed action* is a normal OpenWhisk *action*, but it should accept the following parameters:

- **lifecycleEvent**: one of 'CREATE', 'DELETE', 'PAUSE', or 'UNPAUSE'
- **triggerName**: the fully-qualified name of the trigger which contains events produced from this feed.
- **authKey**: the Basic auth. credentials of the OpenWhisk user who owns the trigger just mentioned

The feed action can also accept any other parameters it needs to manage the feed. For example, the Cloudant changes feed action expects to receive parameters including *'dbname'*, *'username'*, etc.

### *API entity*

This entity allows manifests to link Actions to be made available as HTTP-based API endpoints as supported by the API Gateway service of OpenWhisk.

This entity declaration is intended to provide grammar for the experimental API *(see https://github.com/apache/incubator-openwhisk/blob/master/docs/apigateway.md* and shown using a "book club" example:

#### *Example*

```
$ wsk api create -n "Book Club" /club /books get getBooks
$ wsk api create /club /books post postBooks
$ wsk api create /club /books put putBooks
$ wsk api create /club /books delete deleteBooks
```

the above would translate to the following grammars in the pkg. spec. to a new-top level keyname "apis" in the manifest:

#### *Full Grammar*

```
apis:   # new top-level key for defining groups of named APIs
  <API name>:                 # descriptive name
    <description>: <string>   # optional, description
    basepath: <string>        # optional, shared basepath
    path: <string>            # optional, shared path
    actions:
      <action name>:
         verb: get | post | put | delete
      ...
```

#### *Example: Full Grammar*

```
apis:
  book-club:
    description: Book Club
    basepath: /club
    path: /books
    actions:
      getBooks:
         verb: get
      postBooks:
         verb: post
      putBooks:
         verb: put
      deleteBooks:
         verb: delete
```

### *Simple Grammar*

```
apis:
  <API name>:                    # descriptive name
    description: <string>        # optional, description
    <basepath>:                  # shared basepath
      <path>:
         <action name>: <verb>
         ...
      ...
```

### *Note*

- The 'basepath' and 'path' are required in Simple Grammar; whereas these are optional components in the full grammar.

### *Example: Simplified Grammar*

A somewhat simplified grammar is also supported that allows single-line definition of Actions (names) along with their HTTP verbs.

```
apis:
  book-club:
    club:
      books:
        getBooks: get
        postBooks: post
        putBooks: put
        deleteBooks: delete
      members:
        listMembers: get
```

### *Requirements*

- The API entity's name (i.e., `<API Name>`) MUST be less than or equal to 256 characters.

### *Notes*

- The API entity within the OpenWhisk programming model is considered outside the scope of the Package. This means that API information will not be returned when using the OpenWhisk Package API:
    - `wsk package list <package name>`
- However, it may be obtained using the Trigger API:
    - `wsk api list -v`

## ***Package entity***

The Package entity schema is used to define an OpenWhisk package within a manifest.

### *Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| license | yes | `string` | `unlicensed` | The required value that indicates the type of license the Package is governed by. |

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| credential | no | `string` | `N/A` | The optional Credential used for all entities within the Package. The value is either: Contains either:<br>• A credential string.<br>• The optional name of a credential (e.g., token) that is defined elsewhere. |
| dependencies | no | list of Dependency | N/A | The optional list of external OpenWhisk packages the manifest needs deployed before it can be deployed. |
| repositories | no | list of `Repository` | N/A | The optional list of external repositories that contain functions and other artifacts that can be found by tooling. |
| actions | no | list of `Action` | N/A | Optional list of OpenWhisk Action entity definitions. |
| triggers | no | list of `Trigger` | N/A | Optional list of OpenWhisk Trigger entity definitions. |
| rules | no | list of `Rule` | N/A | Optional list of OpenWhisk Rule entity definitions. |
| feeds | no | list of `Feed` | N/A | Optional list of OpenWhisk Feed entity definitions. |
| compositions | no | list of `Composition` | N/A | Optional list of OpenWhisk Composition entity definitions. |
| apis | no | list of API | N/A | Optional list of API entity definitions. |

### *Requirements*

- The Package `name` MUST be less than or equal to 256 characters.
- The Package entity schema includes all general Entity Schema fields in addition to any fields declared above.
- A valid Package `license` value MUST be one of the Linux SPDX license values; for example: `Apache-2.0` or `GPL-2.0+`, or the value '`unlicensed`'.
- Multiple (mixed) licenses MAY be described using using NPM SPDX license syntax.
- A valid Package entity MUST have one or more valid Actions defined.

### *Notes*

- The Trigger and API entities within the OpenWhisk programming model are considered outside the scope of the Package. This means that Trigger and API information will not be returned when using the OpenWhisk Package API:
  - `wsk package list <package name>`
- However, their information may be retrieved using respectively:
  - `wsk trigger list –v`
  - `wsk api list -v`

### *Grammar*

```
<packageName>:
    <Entity schema> # Common to all OpenWhisk Entities
```

```
    license: <string>
    repositories: <list of repository>
    actions: <list of actions>
    triggers: <list of triggers>
    rules: <list of rules>
    feeds: <list of feeds>
    compositions: <list of compositions>
    apis: <list of API>
```

*Example*

```
my_whisk_package:
  description: A complete package for my awesome action to be deployed
  version: 1.2.0
  license: Apache-2.0
  actions:
    my_awsome_action:
      <Action schema>
  triggers:
    trigger_for_awesome_action:
      <Trigger schema>
  rules:
    rule_for_awesome_action>
      <Rule schema>
```

## Extended Schema

### *Dependencies*

The dependencies section allows you to declare other OpenWhisk packages that your application or project (manifest) are dependent on. A Dependency is used to declare these other packages which deployment tools can use to automate installation of these pre-requisites.

*Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| location | yes | `string` | `N/A` | The required location of the package. |
| version | yes | `version` | `N/A` | The required version of the package. |
| inputs | no | `various` | `N/A` | The optional Inputs to the package. |

*Requirements*

- No additional requirements.

*Notes*

- The ‘version’ parameter is currently used to specify a branch in GitHub and defaults to “master”, this behavior may change in upcoming releases of the specification.

### Grammar

```
<package_name>:
  location: <GitHub URL> |
  version: 1.0.1
  inputs:
    <list of inputs>
```

### Example

```
dependencies:
  status_update:
    location: github.com/myrepo/statusupdate
        version: 1.0.
  database pkg:
    location: /whisk.system/couchdb
    inputs:
      dbname: MyAppsDB
```

## Repository

A repository defines a named external repository which contains (Action) code or other artifacts package processors can access during deployment.

### Fields

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| url | yes | `string` | N/A | Required URL for the Repository. |
| description | no | `string256` | N/A | Optional description for the Repository. |
| credential | no | `string` | N/A | Optional name of a Credential defined in the Package that can be used to access the Repository. |

### Requirements

- The Repository name (i.e., `<repositoryName>` MUST be less than or equal to 256 characters.
- Description string values SHALL be limited to 256 characters.

### Grammar

#### Single-line (no credential)

```
<repositoryName>: <repository_address>
```

#### Multi-line

```
<repositoryName>:
  description: <string256>
  url: <string>
  credential: <Credential>
```

### *Example*

```
my_code_repo:
  description: My project's code repository in GitHub
  url: https://github.com/openwhisk/openwhisk-package-rss
```

## *Credential*

A Credential is used to define credentials used to access network accessible resources. Fields

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| protocol | no | `string` | N/A | Optional protocol name used to indicate the authorization protocol to be used with the Credential's token and other values. |
| tokenType | yes | `string` | password | Required token type used to indicate the type (format) of the token string within the supported types allowed by the protocol. |
| token | yes | `string` | `N/A` | Required token used as a credential for authorization or access to a networked resource. |
| description | no | `string256` | N/A | Optional description for the Credential. |
| keys | no | `map` of `string` | N/A | Optional list of protocol-specific keys or assertions. |

### *Requirements*

- The Credential name (i.e., `<credentialName>` MUST be less than or equal to 256 characters.
- Description string values SHALL be limited to 256 characters.

### *Valid protocol values*

| Protocol Value | Valid Token Type Values | Description |
|---|---|---|
| `plain` | N/A | Basic (plain text) username-password (no standard). |
| `http` | basic_auth | HTTP Basic Authentication Protocol. |
| `xauth` | X-Auth-Token | HTTP Extended Authentication Protocol (base-64 encoded Tokens). |
| `oauth` | bearer | Oauth 2.0 Protocol |
| `ssh` | identifier | SSH Keypair protocol (e.g., as used in OpenStack) |

### *Grammar*

```
Credential:
  type: Object
  properties:
    protocol:
      type: string
      required: false
```

```
    tokenType:
      type: string
      default: password
    token:
      type: string
    keys:
      type: map
      required: false
      entry_schema:
        type: string
    user:
      type: string
      required: false
```

*Notes*

- The use of transparent user names (IDs) or passwords are not considered best practice.

*Examples*

*Plain username-password (no standardized protocol)*

```
inputs:
  my_credential:
    type: Credential
      properties:
        user: my_username
        token: my_password
```

*HTTP Basic access authentication*

```
inputs:
  my_credential:
    type: Credential
    description: Basic auth. where <username>:<password> are a single string
    properties:
      protocol: http
      token_type: basic_auth
      # Note: this would be base64 encoded before transmission by any impl.
      token: myusername:mypassword
```

*X-Auth-Token*

```
inputs:
  my_credential:
    type: Credential
    description: X-Auth-Token, encoded in Base64
    properties:
      protocol: xauth
      token_type: X-Auth-Token
      # token encoded in Base64
      token: 604bbe45ac7143a79e14f3158df67091
```

*OAuth bearer token*

```
inputs:
  my_credential:
    type: Credential
    properties:
      protocol: oauth2
      token_type: bearer
      # token encoded in Base64
      token: 8ao9nE2DEjr1zCsicWMpBC
```

*SSH Keypair*

```
inputs:
  my_ssh_keypair:
    type: Credential
    properties:
      protocol: ssh
      token_type: identifier
      # token is a reference (ID) to an existing keypair (already installed)
      token: <keypair_id>
```

## Package Artifacts

### *Package Manifest File*

The Package Manifest file is the primary OpenWhisk Entity used to describe an OpenWhisk Package and all necessary **schema** and **file** information needed for deployment. It contains the Package entity schema described above.

### *Deployment File*

The Deployment file is used in conjunction with a corresponding Package Manifest file to provide configuration information (e.g., input parameters, authorization credentials, etc.) needed to deploy, configure and run an OpenWhisk Package for a target Cloud environment.
Fields

The manifest and Deployment files are comprised of the following entities:

### *Application*

An optional, top-level key that describes a set of related Packages that together comprise a higher-order application.

*Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| name | no | `string256` | `N/A` | The optional name of the application. |
| namespace | no | `string` | `N/A` | The optional namespace for the application (and default namesapce for its packages where not specified). |

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| credential | no | `string` | | The optional credential for the application (and default credential for its packages where not specified). |
| packages | yes | `list of package` | `N/A` | The required list of one or more packages. |

### *Grammar*

```
application:
  name: <string256>
  namespace: <string>
  credential: <string>
  packages:
    <list of package>
```

### *Requirements*

- The keys under the application key (e.g., name, namespace, credential and packages) are only used in a manifest or deployment file if the optional application key is used.

### *Notes*

- Currently, the OpenWhisk platform does not recognize the Application entity as part of the programming model; it exists as a higher order grouping concept only in this specification. Therefore there is no data stored within OpenWhisk for the Application entity,

### *Example*

```
application:
  name: greetings
  namespace: /mycompany/greetings/
  credential: 1234-5678-90abcdef-0000
  packages:
    helloworld:
      inputs:
        city: Boston
      actions:
        hello:
          inputs: # input bindings
            personName: Paul
    ...
```

### *Notes*

- *Note: A common use would be to associate a namespace (i.e., a target namespace binding) or cedential to a application and all included packages automatically inherit that namespace (if applied at that level) unless otherwise provided (similar to style inheritance in CSS).*

- *The application name would be treated as metadata, perhaps stored in the annotations for the contained entities.*

### *Setting values using ($) dollar notation*

In the Deployment file, a parameter value may be set from the local execution environment by using the dollar ($) notation to denote names of local environment variables which supply the value to be inserted at execution time.

#### *Syntax*

```
<parameter>: $<local environment variable name>
```

#### *Example*

```
...
  inputs:
    userName: $DEFAULT_USERNAME
```

#### *Requirements*

- Processors or tooling that encounter ($) Dollar notation and are unable to locate the value in the execution environment SHOULD resolve the value to be the default value for the type (e.g., an empty string for type string).

#### *Notes*

- Processors or tooling that encounter ($) Dollar notation for values should attempt to locate the corresponding named variables set into the local execution environment (e.g., where the tool was invoked) and assign its value to the named input parameter for the OpenWhisk entity.
- This specification does not currently consider using this notation for other than simple data types (i.e., we support this mechanism for values such as strings, integers, floats, etc.) at this time.

# Normative References

| Tag | Description |
|---|---|
| RFC2119 | S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997. |
| YAML-1.2 | YAML, Version 1.2, 3rd Edition, Patched at 2009-10-01, Oren Ben-Kiki, Clark Evans, Ingy döt Net http://www.yaml.org/spec/1.2/spec.html |
| YAML-TS-1.1 | Timestamp Language-Independent Type for YAML Version 1.1, Working Draft 2005-01-18, http://yaml.org/type/timestamp.html |
| Maven-Version | The version type is defined with the Apache Maven project's policy draft: https://cwiki.apache.org/confluence/display/MAVEN/Version+number+policy |
| OpenAPI-2.0 | The OpenAPI (f.k.a. "Swagger") specification for defining REST APIs as JSON. https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md |
| Linux-SPDX | Linux Foundation, SPDX License list https://spdx.org/licenses/ |
| NPM-SPDX-Syntax | Node Package Manager (NPM) SPDX License Expression Syntax https://www.npmjs.com/package/spdx |

# Non-normative References

| Tag | Description |
|---|---|
| OpenWhisk-API | OpenWhisk REST API which is defined as an OpenAPI document. https://raw.githubusercontent.com/openwhisk/openwhisk/master/core/controller/src/main/resources/whiskswagger.json |
| GNU-units | Size-type units are based upon a subset of those defined by GNU at http://www.gnu.org/software/parted/manual/html_node/unit.html |
| RFC 6838 | Mime Type definitions in compliance with RFC 6838. |
| RFC 7231 | HTTP 1.1. status codes are described in compliance with RFC 7231. |
| IANA-Status-Codes | HTTP Status codes as defined in the IANA Status Code Registry. |
| JSON Schema Specification | The built-in parameter type "json" references this specification. http://json-schema.org/ |

# Scenarios and Use cases

## Usage Scenarios

### *User background*

The following assumptions about the users referenced in the usage scenarios:

- Experienced developer; knows Java, Node, SQL, REST principles and basic DevOps processes; uses IDEs to develop code locally.
- Limited exposure to Serverless, but interested in trying new technologies that might improve productivity.

### *Scenario 1: Clone and Create*

Deploy an OpenWhisk app (project, set of entities, package, ...) discovered on github. The developer...

1. discovers an interesting git repo containing an OpenWhisk app (project, set of entities, package, ...)
2. clones the repo to local disk.
3. He pushes (deploys) it into one of his OpenWhisk namespaces
4. He checks out the app's behavior using OpenWhisk CLI or OpenWhisk UI

#### *Notes*

- while this scenario allows to use the manifest file as a "black box" the manifest format can influence the user experience of a developer trying to read it and understand what it does

### *Scenario 2: Pushing Updates with versioning*

Change a cloned repo that he previously pushed into one of his namespaces. The developer...

1. changes the local repo by editing code and adding and changing entity specifications using local tools (editors, IDEs, ...).
2. bumps version number for package.
3. pushes his updates into the namespace so that the existing entities are changed accordingly.

### *Scenario 3: Start New Repo with Manifest*

Start a new OpenWhisk app (project, set of entities) from scratch. The developer...

1. code files for the actions (e.g. *action1.js, action2.js, action3.js)*
2. creates a LICENSE.*txt file*

3. Creates a **Manifest File** that specifies the set of OpenWhisk entities and their relations (e.g. *manifest.yml).* It also references the LICENSE.txt file.
4. initializes and uploads the set of files as a new git repo.

*Notes:*

- Creating the initial manifest file should be supported by providing an empty template with syntax examples and other helpful comments

### *Scenario 4: Export into Repository*

Share an existing OpenWhisk app (project, set of entities) with others so that they can deploy and change it for their purposes. The developer…

1. exports a defined set of entities (a whole namespace?) into a set of files that includes code files, and generated manifest, LICENSE.txt and README files.
2. initializes and uploads the set of files as a new git repo.
   Example: `git init` … etc.

### *Scenario 5: Discovery and Import from object store*

Discover an OpenWhisk package (manifest) co-located with data in an Object storage service. This package would include a description of the Actions, Triggers, Rules and Event Sources (or Feeds) necessary to interact with data it is associated with directly from the Object storage repository; thus allowing anyone with access to the data an immediate way to interact and use the data via the OpenWhisk Serverless platform.

## Schema Use Cases

### "Hello world"

"Hello world" is a simple use case which will accept a string parameter and return a "Hello world" if the parameter is null or "Hello + <string>" if the string parameter is not null.

### *Manifest Files*

*Example 1: No parameter descriptions*

```
helloworld:
  version: 1.0
  license: Apache-2.0
  actions:
    hello:
      version: 1.0
      function: src/hello/hello.js
      inputs:
        personName: string
```

```
      outputs:
        greeting_string: string
```

### *Defaults discussion*

This specification places an emphasis on simplicity for the casual developer who may wish to hand-code a Manifest and/or Deployment File; however, it also provides a robust optional schema that can be advantaged when integrating with larger application projects using design and development tooling such as IDEs.

In the above example, please note the following intelligent defaults would be applied by a deployment tooling authored using this speification:

- The runtime (i.e., `runtime: nodejs`) is found from the .js extension on the Action function's filename.

### *Example 2: with parameter descriptions*

```
helloworld:
  version: 1.0
  license: Apache-2.0
  actions:
    hello:
      version: 1.0
      function: src/hello/hello.js
      runtime: nodejs@6
      inputs:
        personName:
          type: string
          description: input person name
      outputs:
        greeting_string:
          type: string
          description: output greeting string
```

## Basic Trigger and Rule

This use case will demonstrate how rules and triggers are configured in manifest and deployment file. It will use the previous "helloworld" action, create a rule to associate a trigger with it.

### *Manifest File*

```
helloRules:
  version: 1.0
  license: Apache-2.0
  actions:
    hello:
      version: 1.0
      function: src/hello/hello.js
      runtime: nodejs
      inputs:
        personName: string
      outputs:
```

```
        greeting_string: string

  triggers:
    locationUpdate:
      inputs:
        name: string
        place: string

  rules:
    myRule:
      trigger: locationUpdate
      action: hello
```

### *Deployment File*

```
packages:
  helloRules:
    triggers:
      locationUpdate:
        inputs:
          name: Henry
          place: Washington, D.C.
```

## Github feed

This use case will install a feed to fire a trigger when there is activity in a specified GitHub repository.

### *Manifest File*

```
git_webhook:
  version: 1.0
  license: Apache-2.0
  feeds:
    webhook_feed:
      version: 1.0
      function: github/webhook.js
      runtime: nodejs@6
      inputs:
        username:
          type: string
          description: github username
        repository:
          type: string
          description: url of github repository
        accessToken:
          type: string
          description: GitHub personal access token
        events:
          type: string
          description: the github event type
```

```
  triggers:
     webhook_trigger:
      action: webhook_feed
```

### *Deployment File*

```
packages:
  git_webhook:
    triggers:
      webhook_trigger:
        inputs:
          username: daisy
          repository: https://github.com/openwhisk/wsktool.git
          accessToken:
          events:push
```

## Github feed advanced

This use case use the Github feed to create a trigger. When there is any push event happened, it will send a notification email.

### *Manifest File*

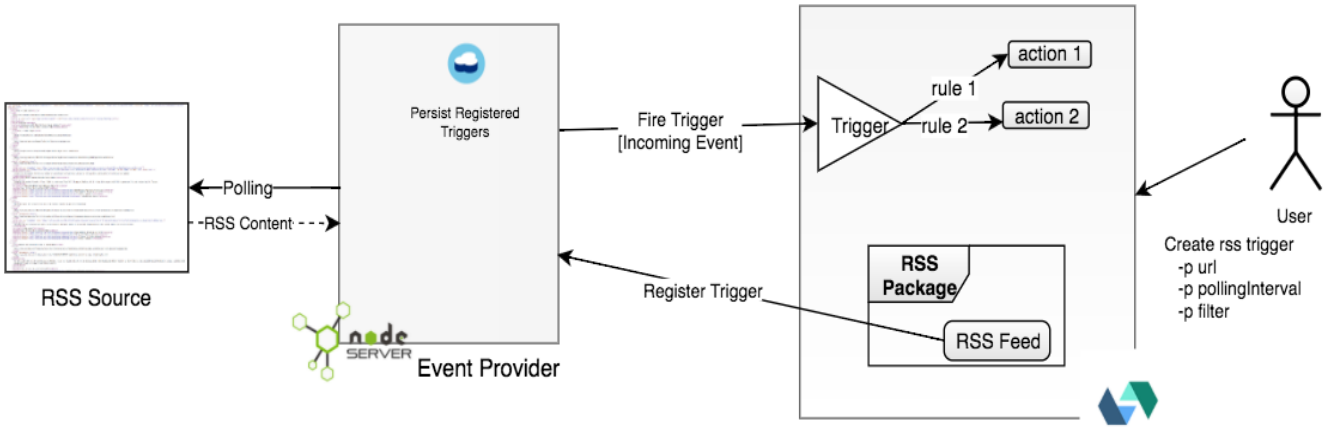
```
git_webhook:
  version: 1.0
  license: Apache-2.0
  action:
    emailNotifier:
      version: 1.0
      function: src/sendemail.js
      runtime: nodejs
      inputs:
        email: string
        title: string
  rules:
    githubNotifier:
      trigger: webhook_trigger
      action: emailNotifier
```

### *Deployment File*

```
packages:
  git_webhook:
    feeds:
      webhook_feed:
        inputs:
          email: daisy@company.com
          title: Github Push Notification
```

## RSS Package

The RSS package provides RSS/ATOM feeds which can receive events when a new feed item is available. It also defines a trigger to listen to a specific RSS feed. It describes the OpenWhisk package reposited here:
https://github.com/openwhisk/openwhisk-package-rss.




### *Manifest File*

#### *with inline values (no Deployment File)*

This example makes use of in-line "values" where the developer does not intend to use a separate Deployment file:

```
rss:
  version: 1.0
  license: Apache-2
  description: RSS Feed package
  inputs:
    provider_endpoint:
      value: http://localhost:8080/rss
      type: string
      description: Feed provider endpoint

  feeds:
    rss_feed:
      version: 1.0
      function: feeds/feed.js
      runtime: nodejs@6
      inputs:
        url:
          type: string
          description: url to RSS feed
          value: http://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml
        pollingInterval:
          type: string
          description: Interval at which polling is performed
          value: 2h
```

```
        filter:
          type: string
          description: Comma separated list of keywords to filter on

  triggers:
    rss_trigger:
      action: rss_feed
```

### *Deployment File*

Alternatively, a Deployment File could have provided the same values (bindings) in this way:

```
packages:
  rss:
    inputs:
      provider_endpoint: http://localhost:8080/rss

    feeds:
      rss_feed:
        inputs:
          url: http://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml
          pollingInterval: 2h
```

Using such a deployment file, allows for more flexibility and the resulting Manifest file would not have needed any '**`value`**' fields.

## Polygon Tracking

This use case describes a microservice composition using Cloudant and a Push Notification service to enable location tracking for a mobile application. The composition uses Cloudant to store polygons that describe regions of interests, and the latest known location of a mobile user. When either the polygon set or location set gets updated, we use the Cloudant Geo capabilities to quickly determine if the new item satisfies a geo query like "is covered by" or "is contained in". If so, a push notification is sent to the user.

### *Manifest File:*

```
application:
  name: PolygonTracking
  namespace: polytrack

  packages:
    polytrack:

      triggers:
        pointUpdate:
          <feed>

        polygonUpdate:
          <feed>
```

```
      actions:
        superpush:
          inputs:
            appId: string
            appSecret: string

        pointGeoQuery:
          inputs:
            username: string
            password: string
            host: string
            dbName: string
            ddoc: string
            iName: string
            relation: string
          outputs:
            cloudantResp: json

        createPushParamsFromPointUpdate:
          <mapper>

        polygonGeoQuery:
          inputs:
            username: string
            password: string
            host: string
            dbName: string
            ddoc: string
            iName: string
            relation: string
          outputs:
            cloudantResp: json

        createPushParamsFromPolygonUpdate:
          <mapper>

      Rules:
        whenPointUpdate:
          trigger:
            pointUpdate
          action:
            handlePointUpdate
        whenPointUpdate:
          trigger:
            polygonUpdate
          action:
            handlePolygonUpdate

      Composition:
        handlePolygonUpdate:
          sequence:
createGeoQueryFromPolygonUpdate,polygonGeoQuery,createPushParamsFromPolygo
nUpdate,superpush
```

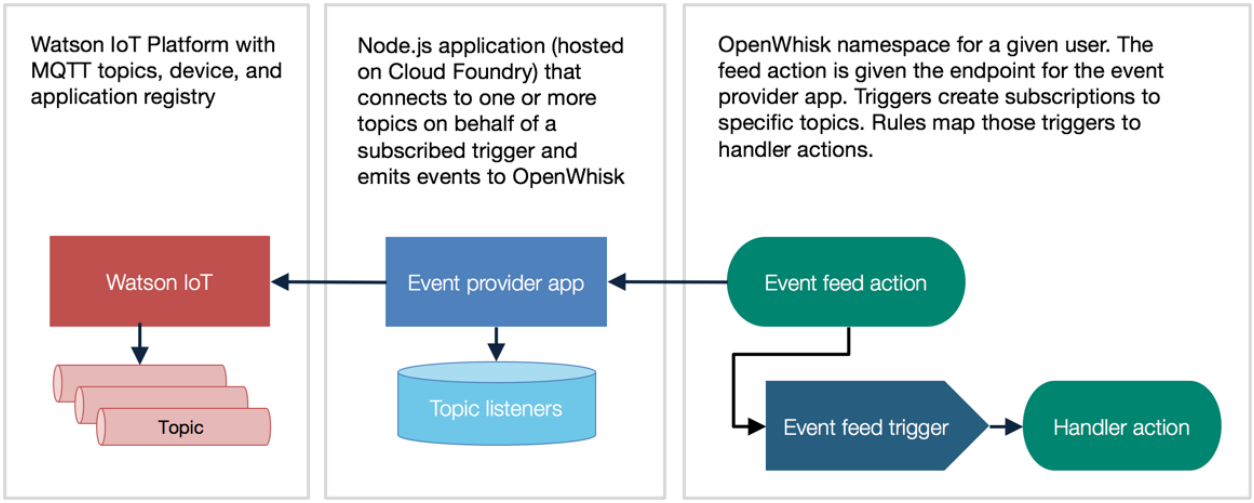
682 ***Deployment File:***

```
application:
  name: PolygonTracking
  namespace: polytrack

  packages:

    myCloudant:
      <bind to Cloudant at whisk.system/Cloudant>

    polytrack:
      credential: ABDCF
      inputs:
        PUSHAPPID=12345
        PUSHAPPSECRET=987654
        COVEREDBY='covered_by'
        COVERS='covers'
        DESIGNDOC='geodd'
        GEOIDX='geoidx'
        CLOUDANT_username=myname
        CLOUDANT_password=mypassword
        CLOUDANT_host=myhost.cloudant.com
        POLYDB=weatherpolygons
        USERLOCDB=userlocation

      triggers:
        pointUpdate:
          <feed>
          inputs:
            dbname: $USERLOCALDB
            includeDoc: true
        polygonUpdate:
          <feed>
          inputs:
            dbname: $USERLOCDB
            includeDoc: true

      actions:
        superpush:
          inputs:
            appId: $PUSHAPPID
            appSecret: $PUSHAPPSECRET
        pointGeoQuery:
          inputs:
            designDoc: $DESIGNDOC
            indexName: $GEOIDX
            relation: $COVEREDBY
            username: $CLOUDANT_username
            password: $CLOUDANT_password
```

```
            host: $CLOUDANT_host
            dbName: $POLYDB
        polygonGeoQuery:
          inputs:
            designDoc: $DESIGNDOC
            indexName: $GEOIDX
            relation: $COVERS
            username: $CLOUDANT_username
            password: $CLOUDANT_password
            host: $CLOUDANT_host
            dbName: $POLYDB
```

## MQTT Package (tailored for Watson IoT)

The MQTT package that integrates with Watson IoT provides message topic feeds which can receive events when a message is published. It also defines a trigger to listen to a specific MQTT topic  It describes the OpenWhisk package reposited here: https://github.com/krook/openwhisk-package-mqtt-watson.




### *Manifest File*

#### *with inline values (no Deployment File)*

This example makes use of in-line "values" where the developer does not intend to use a separate Deployment file:

```
mqtt_watson:
  version: 1.0
  license: Apache-2
  description: MQTT Feed package for Watson IoT
  inputs:
    provider_endpoint:
      value: http://localhost:8080/mqtt-watson
```

```
      type: string
      description: Feed provider endpoint

  feeds:
    mqtt_watson_feed:
      version: 1.0
      function: feeds/feed-action.js
      runtime: nodejs@6
      inputs:
        url:
          type: string
          description: URL to Watson IoT MQTT feed
          value: ssl://a-123xyz.messaging.internetofthings.ibmcloud.com:8883
        topic:
          type: string
          description: Topic subscription
          value: iot-2/type/+/id/+/evt/+/fmt/json
        apiKey:
          type: string
          description: Watson IoT API key
          value: a-123xyz
        apiToken:
          type: string
          description: Watson IoT API token
          value: +-derpbog
        client:
          type: string
          description: Application client id
          value: a:12e45g:mqttapp


  triggers:
    mqtt_watson_trigger:
      action: mqtt_watson_feed
```

### *Deployment File*

Alternatively, a Deployment File could have provided the same values (bindings) in this way:

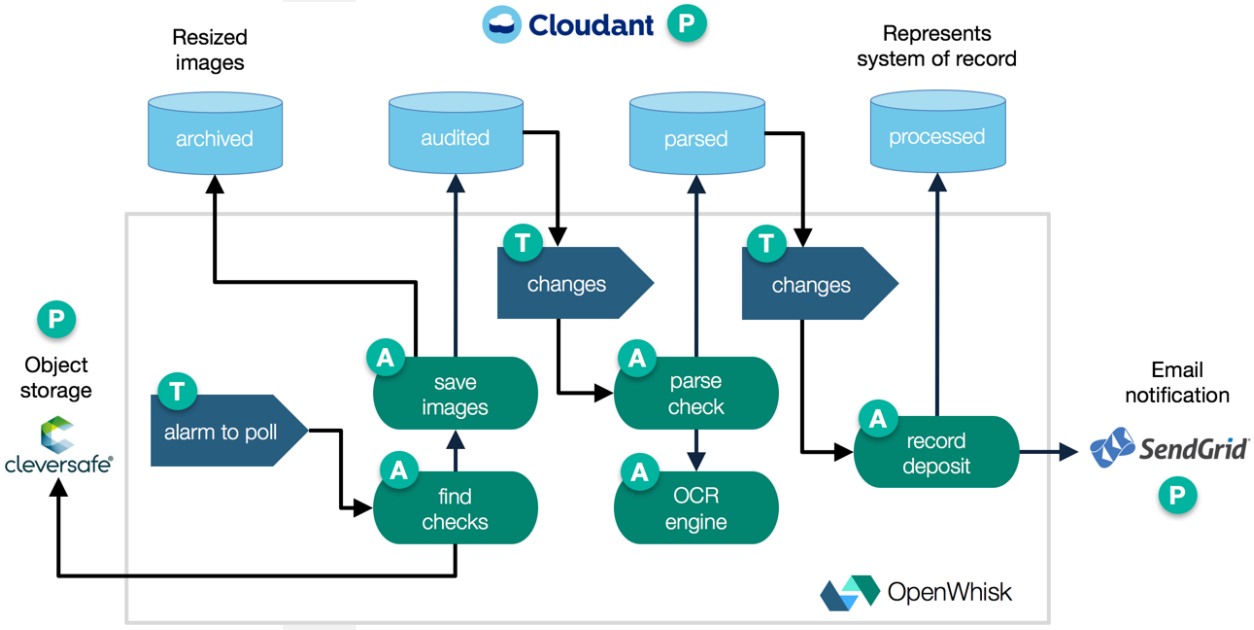
```
packages:
  mqtt_watson:
    inputs:
      provider_endpoint: http://localhost:8080/mqtt-watson

    feeds:
      mqtt_watson_feed:
        inputs:
          url: ssl://a-123xyz.messaging.internetofthings.ibmcloud.com:8883
          topic: iot-2/type/+/id/+/evt/+/fmt/json
          apiKey: a-123xyz
          apiToken: +-derpbog
          client: a:12e45g:mqttapp
```

Using such a deployment file, allows for more flexibility and the resulting Manifest file would not have needed any '**`value`**' fields.

## Check deposit processing with optical character recognition

This use case demonstrates an event-driven architecture that processes the deposit of checks to a bank account using optical character recognition. It relies on Cloudant and SoftLayer Object Storage. On premises, it could use CouchDB and OpenStack Swift. Other storage services could include FileNet or Cleversafe. Tesseract provides the OCR library.

This application uses a set of actions and triggers linked by rules to process images that are added to an object storage service. When new checks are detected a workflow downloads, resizes, archives, and reads the checks then it invokes an external system to handle the transaction.



715 ***Manifest File:***

```
application:
  name: OpenChecks
  namespace: openchecks

  packages:
    openchecks:

    triggers:
      poll-for-incoming-checks:
        inputs:
          cron: string
          maxTriggers: integer

      check-ready-to-scan:
        inputs:
          dbname: string
          includDocs: boolean

      check-ready-for-deposit:
        inputs:
          dbname: string
          includDocs: boolean

    actions:
      find-new-checks:
        inputs:
          CLOUDANT_USER: string
          CLOUDANT_PASS: string
          SWIFT_USER_ID: string
          SWIFT_PASSWORD: string
          SWIFT_PROJECT_ID: string
          SWIFT_REGION_NAME: string
          SWIFT_INCOMING_CONTAINER_NAME: string
          CURRENT_NAMESPACE: string

      save-check-images:
        inputs:
          CLOUDANT_USER: string
          CLOUDANT_PASS: string
          CLOUDANT_ARCHIVED_DATABASE: string
          CLOUDANT_AUDITED_DATABASE: string
          SWIFT_USER_ID: string
          SWIFT_PASSWORD: string
          SWIFT_PROJECT_ID: string
          SWIFT_REGION_NAME: string
          SWIFT_INCOMING_CONTAINER_NAME: string

      parse-check-data:
        inputs:
          CLOUDANT_USER: string
          CLOUDANT_PASS: string
          CLOUDANT_AUDITED_DATABASE: string
          CLOUDANT_PARSED_DATABASE: string
```

```
          CURRENT_NAMESPACE: string

      record-check-deposit:
        inputs:
          CLOUDANT_USER: string
          CLOUDANT_PASS: string
          CLOUDANT_PARSED_DATABASE: string
          CLOUDANT_PROCESSED_DATABASE: string
          CURRENT_NAMESPACE: string
          SENDGRID_API_KEY: string
          SENDGRID_FROM_ADDRESS: string

      parse-check-with-ocr:
        inputs:
          CLOUDANT_USER: string
          CLOUDANT_PASS: string
          CLOUDANT_AUDITED_DATABASE: string
          id: string
        outputs:
          result: JSON

    rules:
      fetch-checks:
        trigger:
          poll-for-incoming-checks
        action:
          find-new-checks
      scan-checks:
        trigger:
          check-ready-to-scan
        action:
          parse-check-data
      deposit-checks:
        trigger:
          check-ready-for-deposit
        action:
          record-check-deposit
```

716 ***Deployment File:***

```
application:
  name: OpenChecks
  namespace: openchecks

  packages:

    myCloudant:
      <bind to Cloudant at whisk.system/Cloudant>

    openchecks:
    credential: ABDCF
    inputs:
      XXX=YYY
```

```
triggers:
  poll-for-incoming-checks:
    <feed>
    inputs:
      cron: */20 * * * * *
      maxTriggers: 90
  check-ready-to-scan:
    <feed>
    inputs:
      dbname: audit
      includeDoc: true
  check-ready-for-deposit:
    <feed>
    inputs:
      dbname: parsed
      includeDoc: true

actions:
  find-new-checks:
    inputs:
      CLOUDANT_USER: 123abc
      CLOUDANT_PASS: 123abc
      SWIFT_USER_ID: 123abc
      SWIFT_PASSWORD: 123abc
      SWIFT_PROJECT_ID: 123abc
      SWIFT_REGION_NAME: northeast
      SWIFT_INCOMING_CONTAINER_NAME: incoming
      CURRENT_NAMESPACE: user_dev
  save-check-images:
    inputs:
      CLOUDANT_USER: 123abc
      CLOUDANT_PASS: 123abc
      CLOUDANT_ARCHIVED_DATABASE: archived
      CLOUDANT_AUDITED_DATABASE: audited
      SWIFT_USER_ID: 123abc
      SWIFT_PASSWORD: 123abc
      SWIFT_PROJECT_ID: 123abc
      SWIFT_REGION_NAME: northeast
      SWIFT_INCOMING_CONTAINER_NAME: container_name
  parse-check-data:
    inputs:
      CLOUDANT_USER: 123abc
      CLOUDANT_PASS: 123abc
      CLOUDANT_AUDITED_DATABASE: audited
      CLOUDANT_PARSED_DATABASE: parsed
      CURRENT_NAMESPACE: user_dev
  record-check-deposit:
    inputs:
```

```
          CLOUDANT_USER: 123abc
          CLOUDANT_PASS: 123abc
          CLOUDANT_PARSED_DATABASE: parsed
          CLOUDANT_PROCESSED_DATABASE: processed
          CURRENT_NAMESPACE: user_dev
          SENDGRID_API_KEY: 123abc
          SENDGRID_FROM_ADDRESS: user@example.org
      parse-check-with-ocr:
        inputs:
          CLOUDANT_USER: 123abc
          CLOUDANT_PASS: 123abc
          CLOUDANT_AUDITED_DATABASE: audited
          id: 123abc
```

# Event Sources

OpenWhisk is designed to work with any Event Source, either directly via published APIs from the Event Source's service or indirectly through Feed services that act as an Event Source on behalf of a service. This section documents some of these Event Sources and/or Feeds using this specification's schema.

## Curated Feeds

The following Feeds are supported by the OpenWhisk platform within the IBM Bluemix Cloud platform. The are referenced by the term "curated" since they are maintained alongside the OpenWhisk open source code to guarantee compatibility. More information on curated feeds can be found here: https://github.com/openwhisk/openwhisk/blob/master/docs/catalog.md.

### Alarms

The `/whisk.system/alarms` package can be used to fire a trigger at a specified frequency. This is useful for setting up recurring jobs or tasks, such as invoking a system backup action every hour.

#### *Package Manifest*

The "alarms" Package Manifest would appear as follows:

```
# shared system package providing the alarms feed action
alarms:
  version: 1.0
  license: Apache-2
  description: Alarms and periodic utility

  actions:
    alarm:
      function: action/alarm.js
      description: Fire trigger when alarm occurs
      feed: true
      inputs:
        package_endpoint:
          type: string
          description: The alarm provider endpoint with port
        cron:
          type: string
          description: UNIX crontab syntax for firing trigger in
Coordinated Universal Time (UTC).
          required: true
        trigger_payload:
          type: object
          description: The payload to pass to the Trigger, varies
          required: false
        maxTriggers:
          type: integer
          default: 1000
          required: false
```

```
feeds:
  location: TBD
  credential: TBD
  operations:
    CREATE:
      TBD
    DELETE:
      TBD
  action: alarm
```

## Cloudant

The `/whisk.system/cloudant` package enables you to work with a Cloudant database. It includes the following actions and feeds.

### *Package Manifest*

The "cloudant" Package Manifest would appear as follows:

```
TBD
```

# Public Sources

The following examples are Event Sources that can provide event data to OpenWhisk. We describe them here using this specification's schema.

## GitHub WebHook

Note: the GitHub WebHook is documented here: https://developer.github.com/webhooks/.

A sample description of the GitHub Event Source and its "create hook" API would appear as follows:

```
TBD
```

# Other Considerations

## Tooling interaction

### Using package manifest directly from GitHub

GitHub is an acknowledged as a popular repository for open source projects which may include OpenWhisk Packages along with code for Actions and Feeds. It is easily envisioned that the Package Manifest will commonly reference GitHub as a source for these artifacts; this specification will consider Github as being covered by the general Catalog use case.

### Using package manifest in archive (e.g., ZIP) file

Compressed packaging, including popular ZIP tools, is a common occurrence for popular distribution of code which we envision will work well with OpenWhisk Packages; however, at this time, there is no formal description of its use or interaction. We leave this for future consideration.

## Simplification of WebHook Integration

### Using RESTify

One possible instance of a lightweight framework to build REST APIs in Nodejs to export WebHook functionality. See https://www.npmjs.com/package/restify
RESTify (over Express) provides help in the areas of versioning, error handling (retry, abort) and content-negotiation. It also provides built in DTrace probes that identify application performance problems.

## Enablement of Debugging for DevOps

### Isolating and debugging "bad" Actions using (local) Docker

Simulate Inputs at time of an Action failure/error condition, isolate it and run it in a "debug" mode.

Considerations include, but are not limited to:

- Isolation on separate "debug" container
- Recreates "inputs" at time of failure
- Possibly recreates message queue state
- Provides additional stacktrace output
- Provides means to enable "debug" trace output
- Connectivity to "other" debug tooling

### Using software debugging (LLDB) frameworks

This is a topic for future use cases and integrations. Specifically, working with LLDB frameworks will be considered. See http://lldb.llvm.org/.

# Appendices

## OpenWhisk Entity Names

### Entity Name Tests

*Note that Java /w matches "word characters" (not just alphanumerics) such as '_' (underscore).*

*Against Java Regex: `\A([\w]|[\w][\w@ .-]*[\w@.-]+)\z`*

| Test String | matches() | n [start(n), end(n)] group(n) | Notes |
|---|---|---|---|
| "a" | true | 0: [0,1] a<br>1: [0,1] a | Shortest Alpha |
| "1" | true | 0: [0,1] 1<br>1: [0,1] 1 | Shortest Numeric |
| "_1" | true | 0: [0,2] _1<br>1: [0,2] _1 | Underscore at beginning |
| "aAbBcCzZ" | true | 0: [0,8] aAbBcCzZ<br>1: [0,8] aAbBcCzZ | All Alphas |
| "1234567890" | true | 0: [0,10] 1234567890<br>1: [0,10] 1234567890 | All Numerics |
| "1 2 a Z 9 0" | true | 0: [0,11] 1 2 a Z 9 0<br>1: [0,11] 1 2 a Z 9 0 | AlphaNumerics with spaces |
| "09 @@ __ 4m .. -- aZ" | true | 0: [0,23] 09 @@ __ 4m .. -- aZ<br>1: [0,23] 09 @@ __ 4m .. -- aZ | Multiples of all possible allowed characters. |
| "Name includes a short domain name is a@b.c" | true | 0: [0,42] Name includes a short domain name is a@b.c<br>1: [0,42] Name includes a short domain name is a@b.c | Sentence-style name with embedded domain name |
| "_" | true | 0: [0,1] _<br>1: [0,1] _ | Underscore only |
| "_." | true | 0: [0,2] _.<br>1: [0,2] _. | Underscore, dot only |
| "_.-@" | true | 0: [0,4] _.-@<br>1: [0,4] _.-@ | Underscore, all special characters only (no spaces) |
| "___" | true | 0: [0,3] ___<br>1: [0,3] ___ | 3 underscores only |
| a+b | false | N/A | Invalid '+' character |
| @a | TBD | | |
| -a | TBD | | |

*Against Java Regex: `\A([\w][\w@ .-]*[\w@.-]+)\z`*

| Test String | matches() | Notes |
|---|---|---|
| TBD | | |

## Acknowledgements

Thanks to the following individuals who have contributed to the contents:

Castro, Paul
Guo, Ying Chun
Krook, Daniel
Linnemeier, Micah
Liu, David
Mitchell, Nick
Ortelt, Thomas
Rutkowski, Matt
Santana, Carlos
Villard, Lionel