HL7_V3_HSP_R1_I1_2019JAN

HL7 Standard:
Health Services Platform (HSP) Marketplace, Release 1

January 2019

HL7 Informative Ballot

Sponsored by:
Service Oriented Architecture Work Group
Implementable Technical Specifications Work Group

IMPORTANT NOTES:

HL7 licenses its standards and select IP free of charge. **If you did not acquire a free license from HL7 for this document,** you are not authorized to access or make any use of it. To obtain a free license, please visit http://www.HL7.org/implement/standards/index.cfm.

If you are the individual that obtained the license for this HL7 Standard, specification or other freely licensed work (in each and every instance "Specified Material"), the following describes the permitted uses of the Material.

A. HL7 INDIVIDUAL, STUDENT AND HEALTH PROFESSIONAL MEMBERS, who register and agree to the terms of HL7's license, are authorized, without additional charge, to read, and to use Specified Material to develop and sell products and services that implement, but do not directly incorporate, the Specified Material in whole or in part without paying license fees to HL7.

INDIVIDUAL, STUDENT AND HEALTH PROFESSIONAL MEMBERS wishing to incorporate additional items of Special Material in whole or part, into products and services, or to enjoy additional authorizations granted to HL7 ORGANIZATIONAL MEMBERS as noted below, must become ORGANIZATIONAL MEMBERS of HL7.

B. HL7 ORGANIZATION MEMBERS, who register and agree to the terms of HL7's License, are authorized, without additional charge, on a perpetual (except as provided for in the full license terms governing the Material), non-exclusive and worldwide basis, the right to (a) download, copy (for internal purposes only) and share this Material with your employees and consultants for study purposes, and (b) utilize the Material for the purpose of developing, making, having made, using, marketing, importing, offering to sell or license, and selling or licensing, and to otherwise distribute, Compliant Products, in all cases subject to the conditions set forth in this Agreement and any relevant patent and other intellectual property rights of third parties (which may include members of HL7). No other license, sublicense, or other rights of any kind are granted under this Agreement.

C. NON-MEMBERS, who register and agree to the terms of HL7's IP policy for Specified Material, are authorized, without additional charge, to read and use the Specified Material for evaluating whether to implement, or in implementing, the Specified Material, and to use Specified Material to develop and sell products and services that implement, but do not directly incorporate, the Specified Material in whole or in part.

NON-MEMBERS wishing to incorporate additional items of Specified Material in whole or part, into products and services, or to enjoy the additional authorizations granted to HL7 ORGANIZATIONAL MEMBERS, as noted above, must become ORGANIZATIONAL MEMBERS of HL7.
Please see http://www.HL7.org/legal/ippolicy.cfm for the full license terms governing the Material.

Ownership. Licensee agrees and acknowledges that **HL7 owns** all right, title, and interest, in and to the Materials. Licensee shall **take no action contrary to, or inconsistent with**, the foregoing.

Licensee agrees and acknowledges that HL7 may not own all right, title, and interest, in and to the Materials and that the Materials may contain and/or reference intellectual property owned by third parties ("Third Party IP"). Acceptance of these License Terms does not grant Licensee any rights with respect to Third Party IP. Licensee alone is responsible for identifying and obtaining any necessary licenses or authorizations to utilize Third Party IP in connection with the Materials or otherwise. Any actions, claims or suits brought by a third party resulting from a breach of any Third Party IP right by the Licensee remains the Licensee's liability.

Following is a non-exhaustive list of third-party terminologies that may require a separate license:

Terminology	Owner/Contact
Current Procedures Terminology (CPT) code set	American Medical Association https://www.ama-assn.org/practice-management/cpt-licensing
SNOMED CT	SNOMED International http://www.snomed.org/snomed-ct/get-snomed-ct or info@ihtsdo.org
Logical Observation Identifiers Names & Codes (LOINC)	Regenstrief Institute
International Classification of Diseases (ICD) codes	World Health Organization (WHO)
NUCC Health Care Provider Taxonomy code set	American Medical Association. Please see www.nucc.org. AMA licensing contact: 312-464-5022 (AMA IP services)

Acknowledgments

The authors wish to recognize the Health Services Platform Consortium (HSPC) for their contributions leading to the publication of this document, making infrastructure resources available for the associated reference implementation of the Marketplace API specification and exemplar web client referenced in Supplementary Reference Implementation.

Portions of this document have been taken from Automated Injection of Curated Knowledge Into Real-Time Clinical Systems: CDS Architecture for the 21st Century by Preston Lee for standardization. The concepts herein are a culmination of works by the authors, HSPC community, and stakeholder organizations.

1 A CALL FOR EXECUTABLE SERVICE PORTABILITY

Services deployed in an enterprise architecture are constituent building blocks in a larger information technology (IT) ecosystem. The deployment and runtime characteristics of each individual building block as well as underlying infrastructure naturally vary greatly across organizations, requiring each service deployment to be further tailored to local IT needs. The process of doing so also varies to a high degree, and is generally considered a normal part of "implementation time" and total cost of ownership (TCO) calculations for incorporating a new service into the enterprise. This lack of infrastructural pre-coordination can result in extremely long IT implementation cycles for any capability to be added to, or changed within, an architecture, especially when underlying infrastructure is completely proprietary to the local institution and/or vendor platform. Burdens of local implementation are confounded by the rise of interest in portable *declarative* clinical knowledge such as HL7 CDS Knowledge Artifacts and Fast Healthcare Interoperability Resources (FHIR) Clinical Reasoning, as well as libraries of directly *executable* clinical knowledge such as HL7 Clinical Quality Language (CQL). Any potential reduction to time spent in discovery, evaluation, update, and other changes to enterprise service oriented architectures (SOAs) – thereby increasing the agility of the organization -- is of great interest to the HIT community. For organizations willing to pre-coordinate choice aspects of service execution itself, direct savings could be substantial. The Marketplace API specification serves as a building block for orchestrating the exchange of such services and executable knowledge.

Newer and upcoming service standards, notably FHIR, FHIR-relevant implementation guidance (e.g. Substitutable Medical Applications Reusable Technologies or "SMART-on-FHIR"), and Clinical Decision Support Hooks (CDS Hooks), present several double-edged swords to health IT (HIT). The ability to decouple standard-based UIs from data authorities allows for greatly flexibility and removes reliance on point-to-point negotiations, thereby allowing application developers to create apps based on common, open APIs.

At the same time, this risks explosive growth in integration complications due to the combinatorics of point-to-point connections required to dynamically offer applications over heterogenous networks. There are many tradeoffs in the continuum between monolithic and microservice architectural principles, and the exponential increase in potential touch points in highly decoupled architectures is not to be discounted. Further, deployment cycles of ISVs rarely align with the maintenance and support capabilities of production environments, often resulting in update delays from years to decades while program managers batch changes from many vendor systems into periodic internal rollouts. Implementations of the Marketplace specification aim to lower these burdens by introducing a capability of service "injection": changing deployed services and knowledge executables within live environments using principles of *continuous deployment* and delivery.

Pertaining to FHIR-oriented specifications such as SMART and CDS Hooks, consideration must be paid to the ability of providers to run traditional on-premise SOA environments. Provider IT groups are already well familiar with the benefits of cloud IaaS, but are also unable to leverage it to boundless degree. A SMART UI, for example, may need to be hosted locally due to VPN, DNS, or content filtering restrictions at the network level. Similarly, a CDS Hooks service may not be permitted to execute over the public Internet as a security policy, lack of suitable service level agreement (SLA), or infinite number of other concerns raised by enterprise IT. In a future full of commoditized HIT services, the ability to exchange service implementations themselves is likely to be critical in scaling the ability to support them.

These benefits, however, are not limited to service specifications. HL7's investment in knowledge representation using CDS Knowledge Artifacts codifies the ability to represent executable queries, but does not venture into the realm of being able to exchange any derived executables (or other runtime artifacts) interoperably. The Marketplace specification aims to provide a critical-path building block for doing so.

2 CORE CONCEPTS

2.1 Summary

The HL7 Health Services Platform Marketplace (HSPM or “Marketplace”) specification is a REST API for publication, cataloging, discovering, and deployment of services and executable knowledge into any compliant IT environment in an automated manner. It is similar to an "app store" for health services in that it manages deployment to a users local infrastructural “Platform” environment and data context. It is *not* a flat directory of SMART-on-FHIR and other UI-only applications, but MAY be used for this purpose. A Marketplace can be implemented by vendors, providers, standards developing organizations (SDOs), consortiums, and all manner of parties interested in interoperable services.

The concept of an application “marketplace” or “app store” is nothing new. This HL7 Marketplace specification, in particular, addresses the problem of exchanging health service implementations themselves in a vendor-neutral manner, which is necessary to move executable artifacts across SOAs with plug-n-play interoperability. A Marketplace *implementation* is a hosted service, operated by any interested party, where executable artifacts are published for exchange and governance policies are defined. This includes backend CDS Hooks implementations, runnable ECA rules/Order Sets/Documentation Templates derived from HL7 CDS Knowledge Artifacts, HL7 CQL, raw FHIR resource servers with known capabilities statements, persistent data stores, and effectively any other service-ized capability that is packaged according to requirements in the Build Packaging section.

2.2 Motivations

The Marketplace specification is not simply a “gallery” designed for automated SMART-on-FHIR (SoF) launch authentication and authorization wiring. SMART-on-FHIR launch addresses the extensions for FHIR servers and OAuth 2 scopes required for access to FHIR resources. SoF only concerns client-side app integration, however, and does not venture into deployment of client applications nor additional backend service: common tasks in enterprise IT. (Note, the Marketplace specification does not intrude on the context synchronization mechanisms of HL7 CCOW and related projects that may depend on OAuth-based infrastructure. Context synchronization and management is out of scope.)

Marketplace design principles are inspired by the business processes that made other computing ecosystems such as the (all proprietary) Apple, Google, and Amazon app stores successful in general-purpose computing, and creates a vehicle for providers to avoid vendor lock-in by adding support for community-managed Marketplace implementations operated as vendor-neutral consortiums and/or credentialing bodies. This approach has been successful in other domains, and the Marketplace API specification is a building block toward replicating those successes in healthcare.

The Marketplace specification also provides an “app store”-like experience for HIT professionals to explore published services, and install them to local or cloud environment with a point-and-click experience similar to that of consumer desktop software and proprietary IaaS. This allows:

- HIT orgs to search for new services across all participating vendors, and deploy them in an automated fashion into on-prem, cloud, and/or hybrid infrastructure, using 1 or more Marketplace instances in any public/private combination.
- Developers to directly submit new (and update existing) Service Builds.

- Marketplace operators to curate, review, and publish vendor Service submissions.
- Compliance validators to automate certification activities and notify deployed Platform environments of critical updates.
- Parties to optionally authenticate with existing SSO credentials shared by other SoF apps/architectures.

An additional selling point of the specification is a complete agnosticism to programming language, frameworks, database, I/O technologies, and most notably, EHR vendor. It permits more architectural flexibility in SMART-on-FHIR clients, CDS Hooks services, and executable knowledge that requires on-site deployment, but does not require implementors to use any of those specifications.

2.3 For Providers and Software Vendors

For an HIT professional interested in *using* a Marketplace, he/she is assumed to operate in a functional architecture where certain business capabilities are present within the target HIT environment and services and executable knowledge will be deployed. This local environment, or Health Services Platform (HSP, "Platform") for short, requires several core capabilities, and is further assumed to integrate with an *ecosystem* of Marketplace implementations. **There is no presumption of HL7, HSPC, EHR vendor, or any other party monopolizing the distribution channels for health services**. If anything, it should be assumed that a local HIT environment with Marketplace integration will support *multiple* public and proprietary Marketplaces each offering a diverse mix of exchanged services.

The Marketplace specification intentionally *omits* two areas of particular interest to vendors that must be addressed by the Marketplace implementer: licensing model enforcement, and sales. It is intended that parties specializing in closed-source, for-profit services will implement "shopping cart"-like features and licensing procedures on top of the Marketplace specification, and that these extensions can be tailed to the operator context. *This specification does not directly facilitate financial transactions.* Publication of commercial software images is highly encouraged, though the acquisition of proprietary licenses is an out-of-band activity between the ISV and consuming party. Commercial software vendors MUST allow limited use of their Service Build images for validation, evaluation, and integration testing purposes prior to achieving a "published" state in the Marketplace and MUST remain so to support automated deployment. In the same vein, Marketplace operators focused on free/open source software (F/OSS) services may elect to omit any further licensing and sales mechanisms entirely.

2.4 For Marketplace Operators and Validators

The Marketplace specification includes a lightweight service curation mechanism. It is assumed that the *operator* of a Marketplace instance does so, in part, as a gateway, certifier, validator, signatory, distributor, or other form of endorser to the services offered for deployment. For example, a FHIR service validation company may integrate with automated test suites to verify builds of a service submission claiming to support US Core profiles does, in fact, behave as expected as part of the build publication workflow.

Compliance and quality assurance organizations may also take interest in the service functional model (SFM) in which a Marketplace implementation is operated, as the curation mechanic is intended to be extended with standard-specific, deep integrations with 3rd-party build validation. Conceptually, this approach creates a standards-based ecosystem for validation by allowing ISVs to independently submit a given Service Build to multiple credentialing organizations without

making vendor-specific concessions regarding the way the executables are packaged. As long as Marketplace Service packaging requirements are met, a byte-identical image may be cross-certified and distributed by unrelated operators simultaneously.

3 FUNCTIONAL INFRASTRUCTURAL MODEL

The Marketplace specification aims to ease service deployment woes in an infrastructure-neutral manner, not complicate them, and acknowledges organizations tend to run services in a combination of:

- locally-provisioned, co-located, and managed VMs using on-premise IaaS tools
- bare metal machines and appliances for one-off use cases and legacy needs
- cloud services such as Amazon Web Service, Microsoft Azure, and Google Cloud

To make automated health Service provisioning possible, all Services MUST meet a set of interoperability criteria prior to being published in the Marketplace that may be validated automatically. The Marketplace API, at the specification level, does not and should not make infrastructure mandates; however, to leverage HL7 Marketplace implementations to the greatest extent, a number of local Platform requirements SHOULD be met. A compatible Health Services Platform conceptually unifies three areas of an IT architecture:

- Packaging - How individual service builds are produced by independent software vendors (ISVs) and consumed by IT groups.
- Curation - Definition and announcement of a service build's capabilities and dependencies.
- Orchestration - Automated service dependency resolution and deployment into the local IT architecture.

3.1 What is a "Health Service"?

The name "Health Services Platform Marketplace" was chosen to imply that a formal definition of a service ("Service") is the focal currency of an implementation. It is thus necessary to provide criteria for what does/does not constitute a Service in the Service Functional Model (SFM) prior to defining the Marketplace API.

Services are declared to the Marketplace via a client such as the reference web UI, other compatible application, or script. Each "Service" instance is primarily a metadata declaration of human-readable fields, and references 0..* *Builds* that in turn point to versioned, executable images. These Builds – or more accurately the remote images they point to – are the meat of what is programmatically bootstrapped into a Platform environment and MAY be subject to additional validation per policies defined by the Marketplace operator, and based on declared capabilities.

3.1.1 Build Packaging

All Service Build images must be:

1. **Containerized** into a single, OCI-compatible image. Docker Community Edition has been used as the gold standard and runtime verification tool and is recommended, but is not required.
2. **Ephemeral**. All persistent data must be saved to an external database and declared as a service dependency, if needed.
3. **Injected with configuration**. All configuration information, including database connection URLs, secret keys, DNS names, base URLs etc MUST be read from runtime

environment variables or autodetected according to Twelve Factor[1] principles. ISVs therefore MUST NOT "bake in" files intended to be manually edited by a system administrator, as no such hand editing is guaranteed in a Platform environment. Configuration variable names MUST ONLY be formed from uppercase A-Z, 0-9 numeric, and underscore '_' characters. Values are always strings. (Lowercase is not permitted due to some environments performing configuration resolution using case-insensitive DNS mechanisms.)

4. **Programmatically validates**. Services declaring standard-compliant capabilities MUST support a mode for exercising declared APIs via a "smoke test" suite that MAY be triggered via a Marketplace itself. Service submissions failing to pass smoke tests on declared capabilities SHOULD be automatically rejected by the operator. Further, ISVs, operators, validators, end users, etc. SHOULD be able to trigger the same smoke test suite if/when they so choose. Operators MAY provide such test harnesses at their discretion. ISV tests specific to a Service or version SHOULD always be run as an out-of-band activity prior to submission to a Marketplace. Marketplace operators MAY elect to charge ISVs for additional testing and validation procedures at the discretion of the operator.
5. **Horizontally scaled**. The number of existing containers SHALL be constrained by the Configuration profile submitted by the vendor, at the discretion of the target Platform within defined constraints.
6. **Dynamically scalable**. Instances are scaled up/down at any time within the limits of the declared Configuration Task minimum/maximum values. Note: For HTTP services, the use of sticky sessions MUST NOT be used, in favor of JWTs or similar lightweight, non-sticky tracking for session state data, when needed.
7. **Single entry process** per Configuration profile Task. If a Service Build requires, for example, the image to be run once as a web service and again as a separate worker node, these alternate entry points should be declared as distinct Tasks within the default Configuration at publication time.
8. **Domain name (DNS) agnostic**. Domain name, SSL/TLS context, and locale-specific settings MUST NOT be hardcoded into a Service Build. Configuration is always either injected at runtime or determined automatically, as can be done with HTTP-based Services.
9. **Unencrypted**. HTTP Services MUST assume that transport encryption is handled at a separate layer. SSL/TLS and reverse proxy load balancing is the responsibility of a target Platform, not the Service Build.
10. **Compute constrained**. Images must define the maximum per-task RAM requirement at image publication time, and manage use of memory internally to prevent exceeding this boundary. Platforms MUST guarantee that these resources will be available, even if overprovisioning virtual environments, and MAY reactively kill Services violating declared constraints.

[1] https://12factor.net

11. **x86-64**. 32-bit binaries are also allowed, but other CPU architectures are not currently supported.
12. **Self-bootstrapped**. Every image MUST be able to bootstrap itself into a functional, default state with zero human intervention. This is assumed at publication time and is used for service validation, local consumer evaluation testing, and for seeding production deployments. Services MUST therefore set up any database schema and perform forward migrations automatically without user intervention at deployment, but MAY operate in limited capacity until out-of-band configuration is performed. This is the intended means for activating proprietary licensing schemes.
13. **Stoppable** at container shutdown time within 10 seconds.
14. **Good citizens** executing in good faith that they do exactly what they say within the environment (e.g. disclosure of operations being performed, data being stored etc), and do not perform other operations that would not be reasonably expected by a Platform operator.
15. **Traceable**. Health via process monitoring internal to the container should support prevailing standards of practice for the applicable software language/framework in use.
16. **Logged**. Services MUST log to standard out/error, and MUST NOT be written to the file system. PHI/PII MAY be logged but SHOULD be toggle-able by the local administrator via injected configuration flags.

Services SHOULD NOT be:

1. **Data payloads**. While nothing prevents using a Marketplace for data distribution, service build images are intended to be used for executable materials. (Consider this a guideline that MAY change in the future.) Large data files SHOULD NOT be bundled into software images. Rather, container initialization steps should be implemented that download requisite data or pull them from a configured database.
2. **Hardware dependent**. This is software that requires specific physical daughter cards, dongles, GPUs, CPU serial numbers etc. Platforms have no assumed means of binding to hardware dependencies. Future extension to image metadata MAY need to make special considerations, however, for frameworks such as OpenCL that aggregate underlying GPU hardware into abstract interfaces. This is currently a matter for future consideration due to differing hardware “passthrough” approaches across environments.

3.1.2 Best Practices

Services SHOULD be:

- SSO Aware using OpenID Connect and/or SAML 2.0. If a service requires user logins, it should be declared as needing an SSO IDP such that configuration can be provided at run-time.
- Provisionable via IETF SCIM 2 API implementations for individual and batch user and group management. This is a computing standard supported by ActiveDirectory and other identity management systems for synchronization of User and Group records.

- Profilable, supporting traceability across the architecture independent of programming language, including tracking of user session context across services for comprehensive system benchmarking (e.g. OpenTracing).
- Incorruptible in the event they are killed without notice.
- Offline-friendly to environments where no Internet access is permitted, or is subject to quality of service disruptions.

3.2 What is a compatible "Platform"?

A Health Services Platform is any infrastructural fabric capable of running service containers packaged distributed according to Marketplace requirements and compatible with the Service Functional Model (SFM). Due to the close relationship between Marketplace functions and Platform capabilities they are two sides of the same coin, though there is no presumption of a "Platform API". While a Platform MAY fully operate without any automated Marketplace integration, and vice versa, doing so limits the potential of automating deployment of knowledge-based (and traditional) services acquired from external parties.

A Platform specification is not a strict specification, per se, but a profile of how to use existing technologies in an interoperable way. This is necessary because enterprise IT environments already have strategic directions on how core virtualization infrastructure is managed, and a prescriptive enterprise architecture strictly prohibiting deviations would not make traction in existing real-world organizations.

3.2.1 Capabilities

A Platform is fundamentally three things. A(n):

1. Cluster of servers capable of running arbitrary Service packages according to Marketplace specifications, at minimum.
2. Orchestration framework of some form for fine-grained management of *running* services after initial deployment.
3. Optional agent synchronizing state changes made by 0..N authorized Marketplaces with the target state of the orchestration controller software.

In the exemplar reference implementation discussed in Supplementary Reference Implementation, these capabilities are provided as follows:

1. A horizontal set of virtual machines running Ubuntu Linux with Docker Community Edition.
2. Out-of-the-box Docker Swarm with Rancher and Portainer.
3. A minimalistic proof-of-concept agent listening for changes to the Marketplace-managed target Platform state, capable of initializing new service instances when an event is sent to the local Agent via push notification.

Note that these supplemental materials are NOT part of the specification and are provided for convenience and further information. An example of fictitious on-premise Platform implementation is illustrated in Figure 1, below.

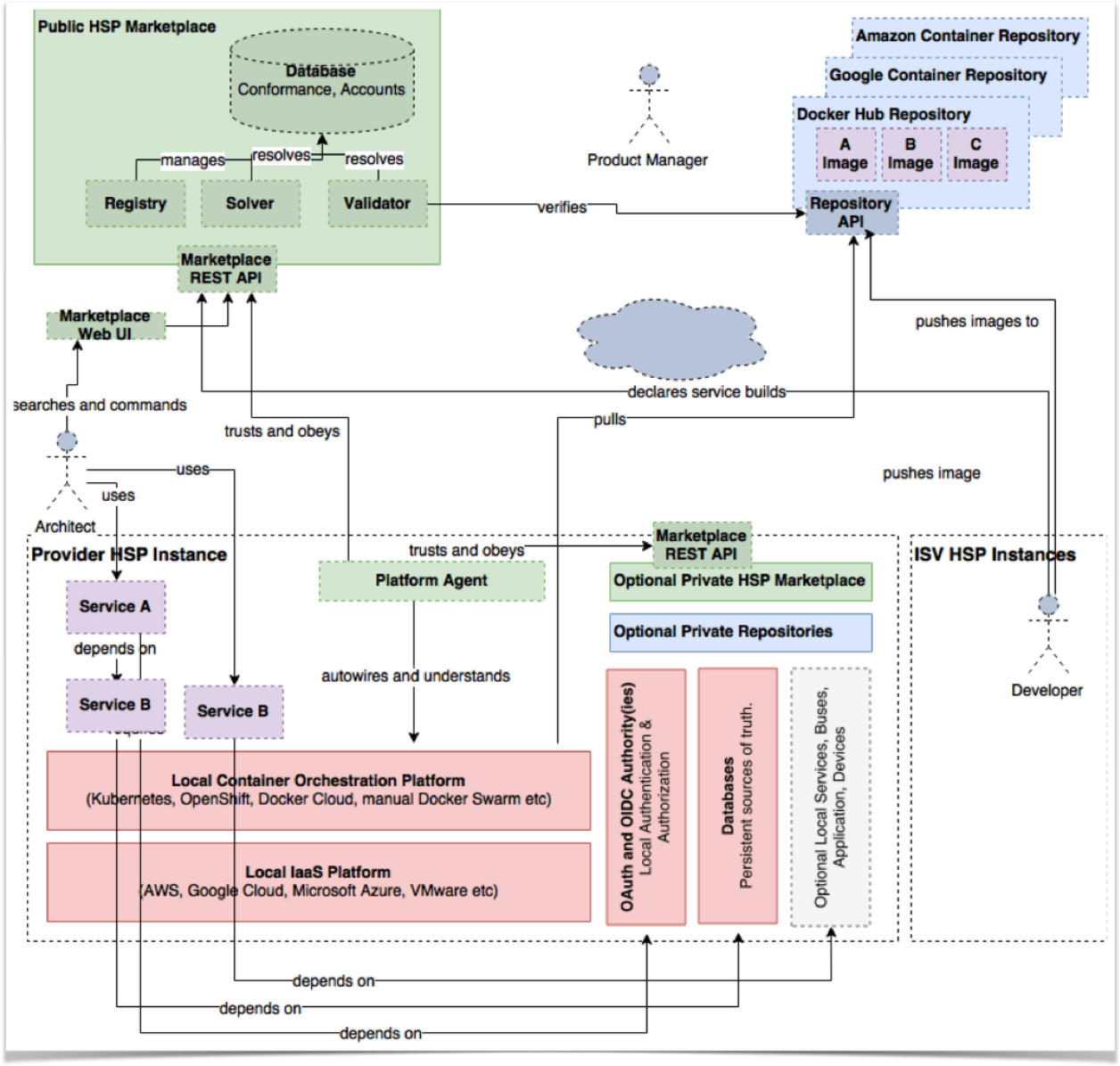

FIGURE 1 EXAMPLE ON-PREMISE PLATFORM ENVIRONMENT.

Note that while the reference implementation uses Docker, it intentionally refrains from using all proprietary functions. Real-world implementations should choose a mainstream management system if not already present.

For Platform environments implemented using cloud resources, Figure 2 illustrates an equally compatible architecture using market-leading Amazon Web Services.[2]

[2] No endorsements or affiliations with Amazon are intended or implied. Example provided for informational purposes only.

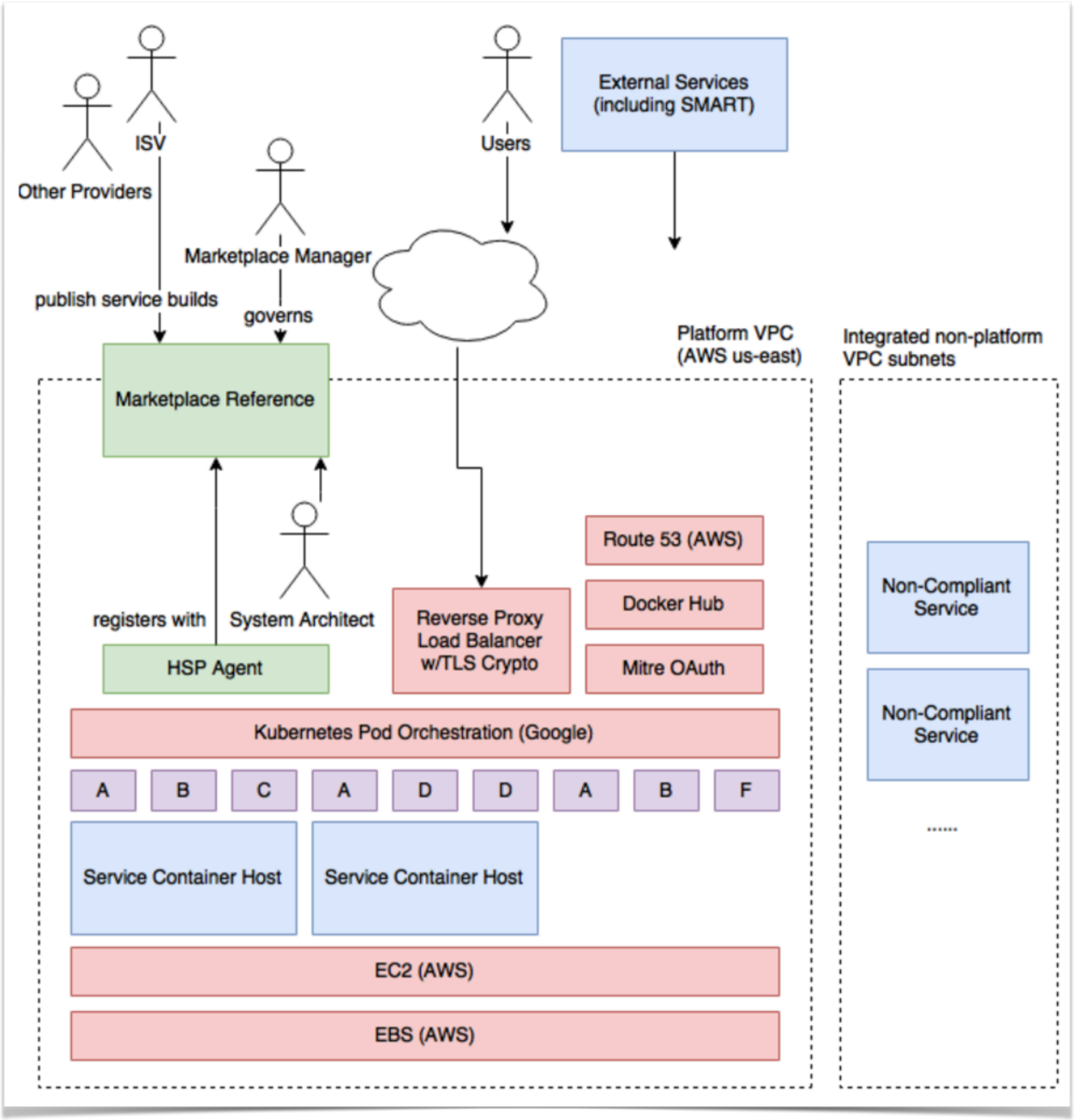

FIGURE 2 CLOUD-BASED PLATFORM IMPLEMENTATION USING AMAZON WEB SERVICES.

3.2.2 Health Services Platform Agent

An Agent is a minimalistic service running on the local Platform listening for state changes in target platform state, as perceived by all configured Marketplace(s) as well as the local orchestration system. The Agent process shown in both Figure 1 and Figure 2 is intended to be minimalistic in nature, only bridging the Marketplace API with the native platform orchestration system. It is completely optional and not part of the Marketplace API specification, but the most significant consumer of the push portion of the API. IT professionals are welcome to use a Marketplace without automated deployment capabilities if desired.

The Agent proof of concept in Supplementary Reference Implementation does very little, only showing how push messages sent from a reference Marketplace may be translated into action by the container platform. More sophisticated deployment profiles are the responsibility of the

orchestration system and Platform operator. In any case, implementers are encouraged to use a stateful system for HSP management to match the stateful nature of the Marketplace's synopsis of local capabilities.

4 API

The Marketplace API is designed as a medium-weight, conventional specification that is straightforward to implement using any number of common-off-the-shelf (COTS) technologies and is easy to understand for simple use cases. Additional considerations have been made for current trends in standards and interoperability.

4.1 Platform Independent Model

The core resources managed by the API are shown in the logical model illustrated in Figure 3, and will be discussed in subsequent subsections.

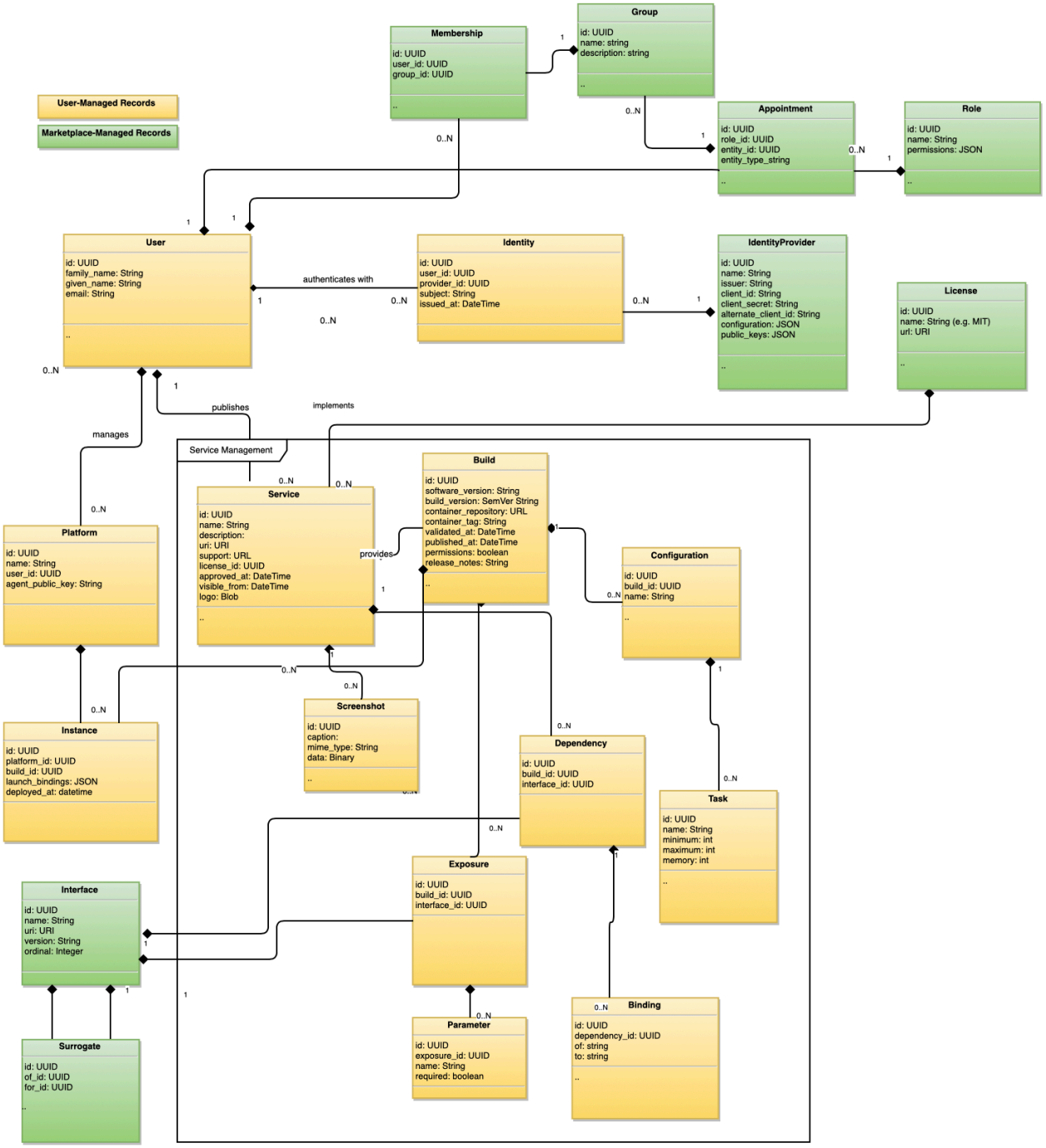

FIGURE 3 PLATFORM INDEPENDANT MODEL (PIM)

4.2 User Identity & Authentication

A Marketplace MUST use OpenID Connect – part of the OAuth 2 family – for Marketplace user authentication using an external single sign-on (SSO) system. Implementations MAY provide support for other SSO protocols/systems including SAML 2.0 and CAS, but OpenID Connect is the required minimum. OAuth 2 enjoys broad mainstream support outside of healthcare, is familiar to enterprise architects, and works well for both mobile and headless clients.

In addition to being a relatively simple specification to implement in most cases, OpenID Connect and OAuth 2 are the basis of SMART-on-FHIR (SoF) authentication and authorization. Marketplace use of OpenID Connect for authentication is not otherwise related to or dependent upon SoF nor FHIR. Likewise, the Marketplace is **not** a FHIR specification and does not provide APIs as FHIR resources. The specific selection of OpenID Connect is intended to enable new and creative means of supporting FHIR-focused Marketplace implementations, when applicable, by providing out-of-the-box compatible authentication against the same external identity provider(s) used for SoF authorization. To reiterate, however, FHIR-based Services or Builds are not required. It is no less valid to operate a Marketplace implementation supporting alternative or unrelated standards. The authentication and authorization objects discussed in subsequent sections are completely unrelated to any similar records that may be present in specific FHIR Services that happen to be published in a Marketplace.

For web-based clients, this implies the use of RFC 7519 JSON Web Tokens (JWTs) within HTTP header bodies to establish the session upon each call. See the JTW/RFC 7519 specification[3] for implementation and usage information.

4.3 Role-Based Access Control

The PIM establishes role-based access control (RBAC) at the User and Group levels using a small number of types. Of particular note is a separation of the User from that of an Identity. **A User is an individual or client actor, whereas an Identity is an issuer-specific credential** establishing authentication information about a User according to the remote authority. Rights are assigned at the Role level via a fine-grained "permissions" field that is recommended (but not required) to be represented internally as JSON to match the JSON format exposed by the Role API. (See Table 1 for an example.)

Each Role MUST provide a permissions object in the form of a JSON associative array, which defaults to an empty "{}". This object MAY grant any number of permissions based on the resource type (noun) and operation (verb) of interest. For example:

```
{
     "services" : {
          "index" : true,
          "read" : true,
          "update" : true,
          "create" : true,
          "delete" : true},
     "builds" : {
          "read" : true},
```

FIGURE 4 ROLE PERMISSIONS TEMPLATE

The permissions object MUST ONLY apply to globally-defined Roles. Implementers MUST allow a User to manage their own Identity, Platform, Instance, and other User-scoped resources

[3] https://jwt.io

without explicit permissions, as well as implicitly grant read access (without explicit permissions) for system objects required to log in via OpenID Connect, such as IdentityProvider. Implementers MAY provide global paths for querying sub-resources without reference to their naturally defined parent resource -- e.g. “GET <root>/platforms” in lieu of “GET <root>/users/:id/platforms” -- and in this case a permission set for “platforms”, “identities” etc would/will apply regardless of the ability to access the parent record.

As a further example of implicit access, a Service MUST be readable, at minimum, by its owner. Management of sub-resources by the owner of the Service, notably Build, SHOULD also be deemed as manageable by the owner, but SHOULD NOT allow limitless control. In the case of Builds – nested within Services as “/services/:uuid/builds” – and other types in that tree, permission to access a sub-resource does not implicitly grant the ability to access the parent, and vice versa. Implementers MAY allow for adaptation, but implementers MUST take care to not inadvertently allow bypass of access controls by querying for related records that include the unauthorized resource.

4.3.1 Administrator Shortcut

A single special global-administrator privilege MUST be supported on the Role permissions field object. This implicitly grants unlimited access to the system for any Role(s) holding it.

```
{
      "manage" : {
            "all" : true
      }
      …
}
```

FIGURE 5 GLOBAL ADMINISTRATION PRIVELEGE

4.3.2 Permission Semantics

Additional permission-level decisions are left to the discretion of the implementer, as are any notion of “built in” or “default” Roles. A Role may be assigned to any combination of User or Group types via a polymorphic Assignment. Permissions across all roles and MUST be unionable and in a “deny-allow” semantic. That is, a User or Group is assumed to **not** hold a permission until/unless it is explicitly granted by an assigned Role. Users/Groups may hold an Assignment to 0..* Roles, and vice versa, and special semantics SHALL NOT be given to the order in which Assignments have been made. A given User or Group SHALL NOT be allowed to hold multiple Assignments to the same Role.

4.3.3 Permission Conflict and Non-Revocation

As the Marketplace RBAC operates on a strict deny-allow semantic, no mechanism is supported to permit "revocation" of privileges or anti-Roles. While defining a mechanism may be tempting to supporting Roles such as "suspended users" or "provisional accounts", doing so would be overly complicated for most implementations. Setting the value of a permission to "false", null, or any other non-true value SHALL NOT have any effect. For example, in cases where a permission is set to true in a Role and false in another, the values SHALL be "or'd" together. In other words, any true value in any Role grants the permission of interest.

4.4 Services, Builds, and Images

A Service is a structured declaration of capabilities for a package of executable or consumable content, with release managed in a discrete lifecycle. A CDS Hooks or FHIR Terminology service "ExampleService" developed by vendor "ExampleSoft", for example, would be declared to a Marketplace instance prior to actual release of the software to establish descriptive text, create screenshots, set standards-related declarations, and other fields.

Builds are concrete, versioned instances of a Service, and is the product versioning mechanism used by the Marketplace. As ExampleSoft provides ongoing development of ExampleService and is ready to release a new version for public deployments, a new Build resource is created under the existing Service declaration with a distinct version within the scope of that Service. Build versions SHOULD adhere to semver[4] semantics. An additional "ordinal" field is provided for Builds that do not adhere to a lexicographically sortable versioning scheme such as hashes or code names. Informal automated Builds made on a recurring basis such as "latest" or "nightly" SHOULD NOT reuse an existing build definition, but MAY so long as this is clearly communicated to the user via other metadata. Distinct Builds are not assumed to be "rolling" or "rebuilt" on a recurring basis.

To ease operational requirements for operating a Marketplace, the Marketplace does not include a mechanism for uploading software. Instead, these Images are declared by reference within the Build record. Images MUST be OCI compliant and be downloadable from the public Internet without special authentication, authorization, or human intervention. (See What is a "Health Service"? for Image packaging requirements.) Implementations MAY implement their own image hosting and management solutions, but this is neither necessary nor required. Numerous general-purpose solutions for management of OCI-compliant images are already available.

4.5 Standards Compliance Declarations

To facilitate automated validation, automated deployment, and autowiring, the Marketplace declares a system-wide collection of standardized Interface types known and supported by the instance. Each Interface MUST have a distinct URI and name, MUST have a version, and MAY have an additional ordinal value. The ability of an Interface to subsume the capabilities of another Interface – for such cases where v2.1 is a superset of all v2.0 functions, for example – is provided via separate Surrogate records. In this example, v2.1 would be allowed to serve as a Surrogate for other Service Builds declaring a Dependency on v2.0.

When ExampleSoft submits a new build of ExampleService, they SHOULD declare adherence to each applicable Interface supported by the Marketplace instance. This declaration is made via 0..* Exposure records subordinate to the Build. Each Exposure thus makes a statement such as, "ExampleService build v1.2.3 provides XYZ API v4.5.6". The exact meaning of this is naturally

[4] https://semver.org

specific to the standard being supported by the Build. Enhancements to the Exposure resource are expected in the future to allow for more robust declaration of highly varied types of Interfaces. Healthcare's widespread use of model "profiling" – a practice far less common in other domains -- makes defining a solution challenging up front. The Interface URI space is intentionally left undefined, though future definition of a global registry may also be warranted. Marketplace operators are encouraged to collaborate on common URIs to avoid Marketplace operator-specific URI declarations.

When a build is deployed, it may set a number of instance-specific Parameters based on the declared Exposures. These are to be used for autowiring purposes, and let local Agents know which settings of the Build, if any, should be visible to other Services/Builds deployed in the future.

Similarly, to support automated deployment, Builds MUST declare their external Service dependencies via Dependency resources if/when they are supported by that Marketplace instance. Dependencies, like Exposures, are specific to each Build, and will likely by refined with additional fields in the future to support more advanced use cases.

The last sub-resource supported by Build is 0..* Configurations. A Configuration contains the runtime and Build entrypoint information essential for a local Agent to execute the Build according to ExampleSoft's runtime requirements. To illustrate, say ExampleService requires a deployment consisting of:

- 2+ containers running a web server
- 1+ background workers
- 1 external database
- 0-1 email servers

For autodeployment to be possible, this Configuration MUST contain a Task resource for each line item. Configurations MUST reference a single Build (and transitively exactly 1 Image). Therefore, Configurations and Tasks MUST ONLY require a single Build to operate. This may change in the future; however, it is not recommend as doing so complicates the entire functional model.

4.6 Configuration State Tracking

One of the more advanced use cases for a Marketplace is fully automated, autowired Service deployment from an external environment into a local platform environment. For a Marketplace to provide intelligent search and filter capabilities for users, it therefore must have some notion of what existing Services are present, regardless of the management system responsible for managing the hosting environment.

Each User MAY declare 0..* Platforms that SHOULD correspond to the environments integrated with that Marketplace instance. Each Platform MUST have a distinct name (relative to that User), which in turn contains 0..* Instance resources. Each Instance refers to the specific Build of the Service running in that Platform environment and contains a JSON field for configuration information set by the Platform agent at runtime. In this initial version of the Marketplace API, there is no ability to define local Services that are *not* otherwise known to the Marketplace. This may change in the future.

4.7 Endpoint Overview

A complete list of endpoints defined by this specification, including all applicable noun/verb combinations, is provided in plaintext format within the accompanying ballot package. This "routes" file is useful for quickly understanding the scope of effort required for implementation and/or consumption.

4.8 Resource Commonalities

Please read and understand this section in entirety prior to the subsequent Endpoints **section. It is required for successful implementation.**

The entire Marketplace API applies a consistent view of RESTful service design with a mission of optimizing ease of consumability for applications developers. A few general principles apply to all resource types.

4.8.1 Endpoint Noun-Verb Paths

For a given resource "foo", paths are always lowercase and plural. Table 1 shows the way resource paths are constructed and the Role permission required to use it. This pattern is repeated for every type of resource unless otherwise noted.

TABLE 1 RESOURCE PATHS AND PERMISSIONS

HTTP Verb	Path	Semantic	Required	Requried Role Permission
GET	/foos	Paginated and filtered index of Foos	Yes	`{"foos" : {"read" : true }}`
POST	/foos	Create a new Foo, automatically assigning a valid UUID if not provided.	Yes	`{"foos" : {"create" : true }}`
GET	/foos/:uuid	Read the specified Foo	Yes	`{"foos" : {"read" : true }}`
PUT or PATCH	/foos/:uuid	Update the specified Foo	Yes	`{"foos" : {"update" : true }}`
DELETE	/foos/:uuid	Delete the specified Foo	Yes	`{"foos" : {"delete" : true }}`
POST	/foos/search	Create a paginated list of Foos functionally beyond "GET /foos"	No	`{"foos" : {"read" : true }}`

For nested resources (aka sub-resources or container resources), the relative path is always appended to the path of the parent it which it is contained. For example, if Foo contains Bar resources, Bars would be located at "/foos/:uuid/bars". Semantically, this relationship MUST imply a specific composition structure binding the lifecycles of the parent and child.

- When the parent is deleted, all children MUST be deleted unless otherwise specified.

- When a child is deleted, the parent MUST NOT be automatically deleted.

4.8.2 UUIDs

All ID types are non-sequential, 128-bit UUIDv4 unless otherwise specified. These are well supported by databases and may also be easily generated using off the shelf libraries available for all popular programming languages. All fields with an "id" in the name are of a UUID type.

4.8.3 Timestamps

All resources instances have "created_at" and "updated_at" datetimes that are maintained by the system. They MUST NOT be modifiable by a client/User, regardless of permissions. These and other datetime types MUST be exposed via the ISO 8601 standard, selected due to globally common use, ease of integration with other systems, and consumability by all common programming languages without custom parsers both inside and outside of healthcare. Any datetime value presented without an explicit time zone MUST be interpreted to be in UTC.

ISO 8601 allows for time zones to be serialized with a datetime string. For datetime fields settable by clients, implementations MUST accept any/all valid embedded time zone values. Regardless of submitted time zone value, implementations SHOULD normalize internally to Universal Time Coordinated (UTC) for database operations and MUST expose them in UTC form back to clients. Clients are responsible for apply any offsets for display purposes based on client-side locale settings. Alternatively, Marketplace implementations MAY provide support for customizable "default" client time zones, but this is discouraged as application of client time zone offsets is generally better handled on the client side and can complicate database implementation operations on the server side.

4.8.4 Paths and URLs

Similar to timestamps, all resource instances also have "path" and "url" fields automatically generated and managed by the server. These fields MUST NOT be modifiable by a client, regardless of permissions. A "path" MUST be a path relative to the detected root URL of the deployed implementation, e.g. "/foos".

A "url" is a full, valid URL of the resource, e.g. "https://marketplace.example.com/foos". The protocol portion of the URL SHOULD match that of the request; implementations MAY force this value to be "https" if automatic http->https redirection is not present in the environmental configuration.

4.8.5 Index Pagination & Filtering

Performing a GET operation at a resource's base path ("/foos") *always* retrieves a paginated index of the resource, subject to Role-Based Access Control. **Pagination support MUST be implemented for all resource indexes.** Filtering is optional.

Pagination and filtering are controlled using two query parameters:

TABLE 2 INDEX PAGINATION AND FILTER PARAMETERS

Parameter	Default	Constraints	Example
page	1	1-based positive integer	42
per_page	10	Positive integer 1 or greater	50
<field_name>	ignored	MUST be coercible into the correct type. Non-strings MUST	name=preston

		be matched exactly. Strings SHOULD be fuzzy matched.	version=1.0.0
sort	<field_name>	Must be a valid, accessible field if provided. Otherwise, implementations may select the default field for sorting, if any.	name
order	undefined	Must be “ascending” or “descending” if present. Otherwise, implementation may provide any ordering.	ascending

Implementers MAY allow pageless “dumps” of a resource by providing a per_page of 0, though this is not recommended for types expected to have a large number of underlying collections to avoid “N+1” and similar performance issues.

The response to an index operation MUST follow the below template, created to provide the easiest possible path for client developers to navigate the paginated results. This is based on real-world usage within hundreds of existing applications.

```
{
    "total_pages": 1,
    "total_entries": 2,
    "previous_page": null,
    "next_page": null,
    "current_page": 1,
    "results": [
          <actual resources>
     ]
}
```

FIGURE 6 FILTERED INDEX RESPONSE TEMPLATE

4.8.6 Search

Performing a GET operation at a resource’s base path (“/foos”) retrieves a filtered index according to Index Pagination & Filtering. Implementors MAY provide an additional “POST /foos/search” with the same parameters if POSTing is required by client code.

4.9 Special Endpoints

Several one-off endpoints differ from the conventions defined in Resource Commonalities. These are listed below.

HTTP Verb	Path	Purpose	Template
GET	/	Root path	`{"message": "This service provides an API only and does not offer a built-in graphical interface."}`
GET	/status	System availability and health.	`{ "message": "This application server and underlying database connection appear to be healthy.", "service": { "datetime": "2018-11-27T21:30:05.967-07:00" }, "database": { "datetime": "2018-11-28T04:30:05.967+00:00" } }`
GET	/sessions	OAuth 2 callback after authentication	Redirect to client URL with `"jwt=…"` parameter or return JWT: `{"jwt": "…", "authorization": "Bearer …"}`
POST	/session	Begin OAuth2 authentication for given provider_id	Redirect to IDP corresponding to `provider_id` (See additional option in)
DELETE	/session	Invalidates given JWT authentication header	`{"message": "Logged out."}`

4.10 Endpoints

Resource templates and resource-specific notes for all supported types are provided in the following sections. Please read sections on Resource Commonalities prior to endpoint-specific sections, as only resource-specific data structures are covered here.

4.10.1 IdentityProviders (/identity_providers)

An IdentityProvider is a deployment-specific resource containing client configuration information for a resource OpenID Connect authentication/authorization server. This information changes very infrequently.

Of special note is the "public_keys" field, which SHOULD be polled and updated automatically. Issuers typically cycle through keys pairs frequently and failing to update them will result in failed User authentication flows.

```
{
      "id": "066dc94a-caed-421a-866d-0bfe117dd0a0",
      "name": "Google", // Distinct
      "issuer": "https://accounts.google.com", // Distinct
      "redirect_url":
"http://localhost:3000/identity_providers/066dc94a-caed-421a-866d-
0bfe117dd0a0/redirect.json",
      "client_id": "…", // Provided by the issuer
      "alternate_client_id": "…",
      "client_secret": "…",
      "scopes": "…", // List of OAuth scopes
      "configuration": {}, // OAuth data programmatically provided
      "public_keys": {}, // Rotated by the issuer
      "enabled_at": "2018-11-27T22:46:06.609Z", // Null if disabled
      "path": "/identity_providers/066dc94a-caed-421a-866d-
      0bfe117dd0a0.json",
      "url": http://localhost:3000/identity_providers/066dc94a-caed-
      421a-866d-0bfe117dd0a0.json,
      "created_at": "2018-11-27T22:46:06.609Z",
      "updated_at": "2018-11-27T22:46:06.609Z",
}
```

4.10.2 Users (/users)

Users resources represent the individuals or applications making Marketplace API requests.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "external_id": "…", // For SCIM and IAM integration
      "name": "Administrator", // Required
      "first_name": "…",
      "middle_name": "…",
      "last_name": "…",
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/users/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911",
      "path": "/users/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.3 User Identities (/users/:id/identities)

An Identity contains the IdentityProvider-specific information for a given User that has authenticated against the supported IdentityProvider. Most of the fields are provided from the IdentityProvider during authentication.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "user_id": "<given user id>", // Required
      "identity_provider_id": "configured idp id", // Required
      "sub": "…", // Required from OAuth
      "iat": "…", // From OAuth
      "hd ": "…", // From OAuth
      "locale ": "…", // From OAuth
      "email": "…", // From OAuth
      "hd ": "…", // From OAuth
      "notify_via_email ": true, // Required, default true
      "notify_via_sms": false, // Required, default false
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/users/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/identities/50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "path": "/users/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/identities/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
```

4.10.4 User Platforms (/users/:id/platforms)

A Platform is a User declaration of a compatible local runtime environment, possibly managed by an automated Agent.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "user_id": "<given user id>", // Required
      "name": "My Test Platform", // Required, distinct for the
     user
      "public_key ": "…", // Provided by agent
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/users/50d5ecf2-1b24-4adb-
     98be-2f6fabe2b911/platforms/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911",
      "path": "/users/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/platforms/50d5ecf2-1b24-4adb-98be-
```

4.10.5 User Platforms (Service) Instances (/users/:id/platforms/:id/instances)

A Service Instance declares a running instance of a known Build on a specific User Platform.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "platform_id": "<given platform id>", // Required
      "build_id": "<given build id>", // Required
      "launch_bindings": {}, // Required, config object
      "deployed_at": "2018-11-27T22:38:49.666Z ", // Set by
     Agent if/when running
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/users/50d5ecf2-1b24-4adb-
     98be-2f6fabe2b911/platforms/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/instances/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911",
      "path": "/users/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/platforms/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/instances/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911"
}
```

4.10.6 Groups (/groups)

A Group allows for batch assignment of Roles to a collection of Users.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "name": "CDS Team", // Required, distinct
      "description": "…", // Required, distinct
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/groups/50d5ecf2-1b24-4adb-
     98be-2f6fabe2b911",
      "path": "/groups/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.7 Group Members (/groups/:id/members)

A Member resource assigns a given user to a group.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "user_id": "…", // Required, distinct per group_id
      "group_id": "…", // Required, distinct per user_id
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/groups/50d5ecf2-1b24-4adb-
     98be-2f6fabe2b911/members/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911",
      "path": "/groups/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/members/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.8 Roles (/roles)

A Role is a declaration of a fine-grained set of permissions. The “default” Boolean field MUST specify whether or not the Role will be automatically Appointed to new User/Group resources from this point in time at which it is set to true and forward.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "name": "Managers", // Required, distinct
      "description": "…", // Required, distinct
      "default": false, // Required, default false.
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/roles/50d5ecf2-1b24-4adb-
     98be-2f6fabe2b911",
      "path": "/roles/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.9 Role Appointments (/roles/:id/appointments)

A Role Appointment is a polymorphic type assigning a Role to single User or Group.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "role_id": "…", // Required
      "entity_id": "…", // Required, User or Group id
      "entity_type": "…", // Required: "User" or "Group"
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/roles/50d5ecf2-1b24-4adb-
     98be-2f6fabe2b911/appointments/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911",
      "path": "/roles/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/appointments/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911"
```

4.10.10 JsonWebToken (/json_web_tokens)

A JsonWebToken is a stateful session identifying a User through an authorized Identity. Endpoints for the JsonWebToken type are purely OPTIONAL and SHALL NOT be required for API usage but may be useful for clients focused on operational tasks such as invaliding sessions of all currently-authenticated users. If present, they SHALL be exposed according to the same conventions of other resources.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "identity_id": "…", // Required User Identity
      "expires_at": "2018-11-27T22:38:49.666Z", // Required
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/json_web_tokens/50d5ecf2-
    1b24-4adb-98be-2f6fabe2b911",
      "path": "/json_web_tokens/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911"
}
```

4.10.11 Licenses (/licenses)

Licenses are globally declared terms of use, and MAY or MAY NOT be useful in all Marketplace operator contexts. As discussed in the For Providers and Software Vendors section, this is expected to be customized to the local sales and business model. Implementors and operators only focused on F/OSS Services MAY be satisfied by the barebones model.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "name": "…", // Required, distinct
      "url": "2018-11-27T22:38:49.666Z", // Required, distinct
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/licenses/50d5ecf2-1b24-
    4adb-98be-2f6fabe2b911",
      "path": "/licenses/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.12 Services (/services)

Service declarations contain the core metadata of a deployable package independent of any specific Build or other versioned release.

The logo_* fields are all set by the server automatically whenever “logo” binary field data are submitted to the server via a POST, PUT or PATCH. These generated fields SHOULD NOT be modifiable by the User/client directly.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "user_id": "…", // Required User owner
      "license _id": "…", // Required, License needed to use
      "name": "…", // Required, distinct
      "description": "…", // Required, distinct
      "uri": "…", // Required identifier of the product offering
     across all versions
      "published_at": "2018-11-27T22:38:49.666Z", // Set by site
     operator, must be set to be discoverable
      "visable_at": "2018-11-27T22:38:49.666Z", // Set by User
     owner, must be set to be discoverable
      "logo_file_name": "custom_logo.jpg",
      "logo_file_size": 29382, // Size of original, in bytes
      "logo_content_type": "image/jpeg",
      "logo_updated_at": "2018-11-27T22:38:49.666Z",
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/services/50d5ecf2-1b24-4adb-
     98be-2f6fabe2b911",
      "path": "/services/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

In addition to the conventional REST endpoints, several others are made available for logo rendering. Whether or not CORS must be enabled for these endpoints is left to the discretion of the implementor and operator, though it is recommend to allow CORS for their official clients, at minimum. MIME types returned by the server MUST match the content of the binary data.

TABLE 3 ADDITIONAL SERVICES ENDPOINTS

Verb	Path	Returns
GET	/services/:id/small_logo	100x100 binary
GET	/services/:id/medium_logo	400x400 binary
GET	/services/:id/large_logo	1600x1600 binary
POST	/services/:id/publish	Resource JSON with published_at set to current datetime.
POST	/services/:id/publish	Resource JSON with published_at set to null.

4.10.13 Service Screenshots (/services/:id/screenshots)

Screenshots are optional images provided by a Service developer previewing select aspects of the Service. For cases where a Service provides no visual components or GUI, it is suggested that developers SHOULD still submit images previewing meaningful interactions on the Exposed Interface(s).

The binary image data associated with a screenshot is handled similarly to the Service logo and derivative fields: image_* fields are all set by the server automatically whenever "image" binary field data is submitted to the via a POST, PUT or PATCH. These generated fields SHOULD NOT be modifiable by the User/client directly.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "service_id": "…", // Required Service
      "license_id": "…", // Required, License needed to use
      "caption": "…", // Required
      "image_file_name": "screenie1.jpg",
      "image_file_size": 29382, // In bytes
      "image_content_type": "image/jpeg",
      "image_updated_at": "2018-11-27T22:38:49.666Z",
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/services/50d5ecf2-1b24-4adb-
    98be-2f6fabe2b911/screenshots/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911",
      "path": "/services/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/screenshots/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

Like the Service logo endpoints, Screenshots also provide direct binary access resources for direct client rendering. Whether or not CORS must be enabled for these endpoints is left to the discretion of the implementor and operator. MIME types returned by the server MUST match the content of the binary data.

TABLE 4 ADDITIONAL SERVICE SCREENSHOT ENDPOINTS

Verb	Path	Returns
GET	/services/:id/screenshots/:id/small	100x binary (various height)
GET	/services/:id/ screenshots/:id/medium	400x binary (various height)
GET	/services/:id/ screenshots/:id/large	1600x binary (various height)

4.10.14 Service Builds (/services/:id/builds)

A Build is a specific version release of a given Service. See Services, Builds, and Images.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "service_id": "…", // Required Service
      "service_version": "…", // Required, distinct per Service
      "version": "…", // Required, semver is preferred
      "release_notes": "…", // Required
      "container_repository": "…", // Required container registry
      "container_tag": "…", // Required
      "published_at": "2018-11-27T22:38:49.666Z", // Set by site
     operator, must be set to be discoverable
      "validated_at": "2018-11-27T22:38:49.666Z", // Set by site
     operator, must be set to be discoverable
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/services/50d5ecf2-1b24-4adb-
     98be-2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911",
      "path": "/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.15 Service Build Dependencies (/services/:id/builds/:id/dependencies)

A Build Dependency is a runtime requirement of a given Service. See Services, Builds, and Images.

The “required” field defines whether or not the referenced Interface is mandatory at runtime. The “mappings” object is used for key/value pairs defining how an exposed parameter is mapped into the configuration of the referenced build at runtime. This capability is expected to expand significantly in future versions.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "build_id": "…", // Required Build
      "interface_id": "…", // Required Interface, distinct per Build
      "required": "…", // Required, default true.
      "mappings":{}, // Required Interface, distinct per Build
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/dependencies/50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "path": "/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-2f6fabe2b911/
     dependencies/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.16 Service Build Exposures (/services/:id/builds/:id/exposures)

A Build Exposure declares that a given Service Build provides capabilities required of a known Interface. See Services, Builds, and Images.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "build_id": "…", // Required Build
      "interface_id": "…", // Required Interface, distinct per Build
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/exposures/50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "path": "/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/exposures/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.17 Service Build Exposure Parameters (/services/:id/builds/:id/exposures/:id/parameters)

A Build Exposure Parameter declares that a given Service Build Exposure (of an Interface) defines a named configuration parameter at runtime, and whether or not the Parameter is required to successfully expose the Interface. See Services, Builds, and Images.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "exposure_id": "…", // Required Exposure
      "name": "…", // Required, distinct per Exposure
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/services/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/exposures/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/parameters/50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "path": "/services/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/exposures/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/parameters/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.18 Service Build Configurations (/services/:id/builds/:id/configurations)

A Service Build Configuration outlines the deployment profile of a running Instance in initial operating capacity. See Services, Builds, and Images.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "build_id": "…", // Required Build
      "name": "…", // Required, distinct per Build
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/configurations/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911",
      "path": "/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/configurations/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911"
}
```

4.10.19 Service Build Configuration Tasks (/services/:id/builds/:id/configurations/:id/tasks)

A Service Build Configuration Task defines a container-based command needed to run the Configuration of the (transitively referenced) Build. See Services, Builds, and Images.


```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "configuration_id": "…", // Required Configuration
      "name": "…", // Required, distinct per Configuration
      "command": null, // Entrypoint command string, default null. Null
     results in the built-in default within the Build image
      "minimum": 2, // Min instances at runtime. Required. Must be >= 1.
      "maximum": 8, // Max instances at runtime. Required. Must be >=
     minimum. 0 for unlimited.
      "memory": 1024, // Required, in MiB
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/configurations/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/tasks/50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "path": "/services/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/builds/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/configurations/50d5ecf2-1b24-4adb-98be-
     2f6fabe2b911/tasks/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.20 Interfaces (/interfaces)

An Interface declares system-wide knowledge of a standardized – or the least conventionalized – computational interface. See Standards Compliance Declarations.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "name": "…", // Required, distinct
      "uri": "…", // Required, distinct
      "version": "…", // Required.
      "ordinal": 42, // Required, default 0. For display ordering
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/interfaces/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911",
      "path": "/interfaces/50d5ecf2-1b24-4adb-98be-2f6fabe2b911"
}
```

4.10.21 Interface Surrogates (/services/:id/surrogates)

An Interface Surrogate declares that the reference substitute provides compatible capabilities of the given Interface. See Standards Compliance Declarations.

```
{
      "id": "50d5ecf2-1b24-4adb-98be-2f6fabe2b911",
      "interface_id": "…", // Required
      "substitute_id": "…", // Required, distinct per interface_id
      "created_at": "2018-11-27T22:38:49.666Z",
      "updated_at": "2018-11-27T22:38:49.666Z",
      "url": "http://localhost:3000/interfaces/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/surrogates/50d5ecf2-1b24-4adb-98be-2f6fabe2b911",",
      "path": "/interfaces/50d5ecf2-1b24-4adb-98be-
    2f6fabe2b911/surrogates/50d5ecf2-1b24-4adb-98be-2f6fabe2b911","
}
```

4.10.22 WebSockets (/websockets)

The WebSockets interface is an experimental bidirectional TCP channel established between a client, such as an Agent, to receive push notifications around Marketplace activity. This is an OPTIONAL feature and no strict protocol exists at this time for implementation. This area of the specification is expected to expand greatly in future revisions to standardize the message format, subscription mechanism, and scope of function.

See notes in Marketplace Service for starting points and exploration of the proof-of-concept pub/sub mechanism used by external reference materials.

5 SUPPLEMENTARY REFERENCE IMPLEMENTATION

In drafting this specification, several reference implemention projects were created as a means of vetting core use cases and modeling decisions. None of these projects should be considered part of the specification proper, but are useful in evaluation and planning. All projects discussed in this section have been made available under unencumbered F/OSS license and may be forked with or without explicit permission. They are subject to change without notice and are expected to do so from ongoing feedback.

5.1 Marketplace Service

The conceptual demonstration of the API implementation defined in the API section, including a basic bi-directional WebSocket mechanism for Agent automation, is available under Open Source license, here: https://github.com/preston/hsp-marketplace-server . At time of this writing a demo environment is available by request from HSPC.

The reference implementation provides platform-specific model (PSM) for PostgreSQL illustrated in Figure 7 Reference Marketplace Service Platform-Specific Model, and itself is built to be distributable as a Health Service by an instance of itself. The demo further demonstrates the principle of self-bootstrapping an underlying, external database into the correct state without human intervention, and implements 12-Factor configuration injection and other principles required of all Health Services.

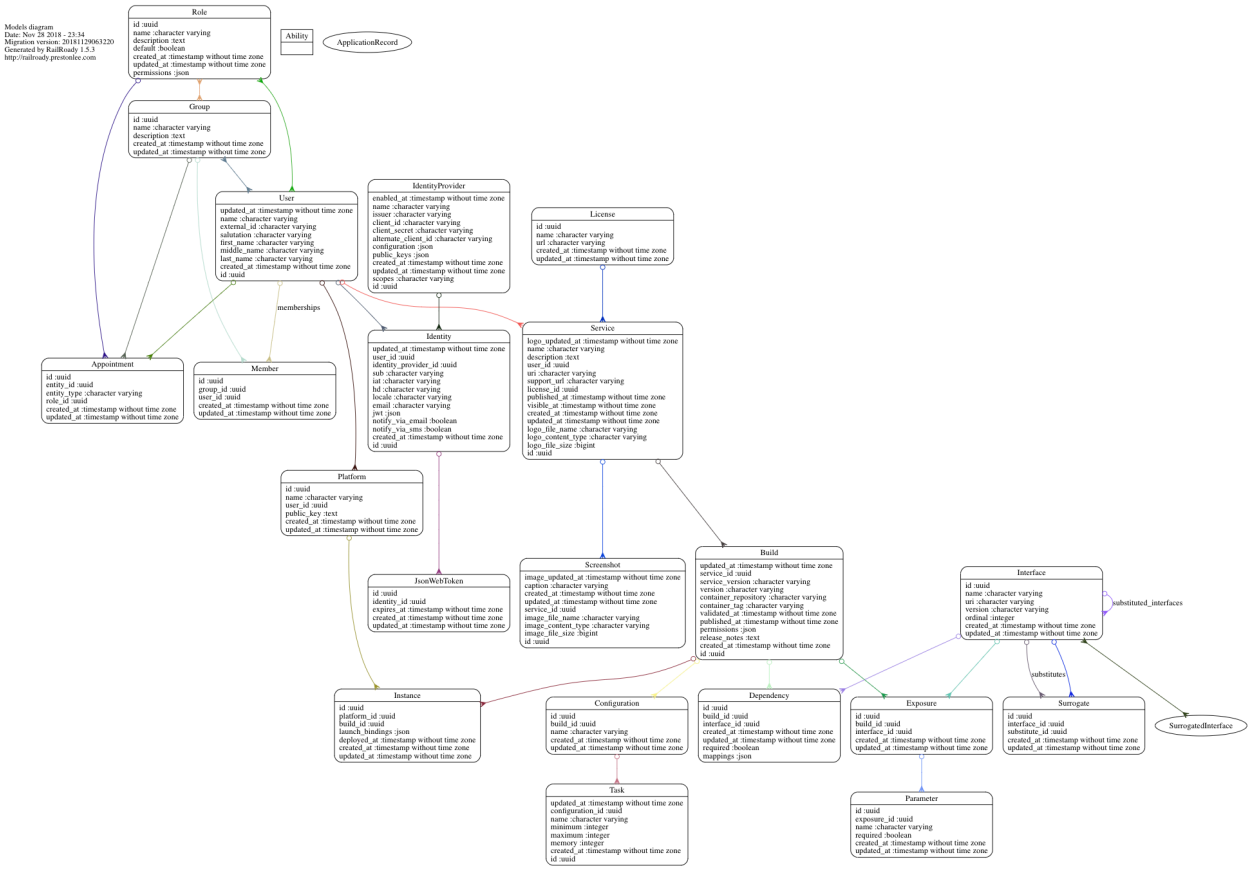

FIGURE 7 REFERENCE MARKETPLACE SERVICE PLATFORM-SPECIFIC MODEL

The reference implementation further provides an integrated WebSockets-based pub/sub mechanism at the /websockets endpoint, implemented using the space_elevator library (https://github.com/preston/space_elevator) to receive Platform-instance-specific messages over a stateful Marketplace server connection. This demonstrates support for the automated point-and-click deployments ultimately envisioned.

5.2 Marketplace UI

While the Marketplace API does not define or assume any GUI capabilities, it is often easier to illustrate the capabilities of the Marketplace's functional model with a breathing, interactive example.

The reference Marketplace UI is a web application written to the Marketplace API, using modern web technologies and in mainstream development style. It authenticates using OAuth 2 and OpenID Connect in the same manner as SMART-on-FHIR, and shows how frontend-only web applications may be easily containerized to meet Build Packaging requirements. Like the reference Marketplace Service, it is also packaged to be distributed by a Marketplace instance. Source code is available, here:

- https://github.com/hspc/hsp-marketplace-server
- https://github.com/hspc/hsp-marketplace-ui

A demo environment is available at HSPC by request.

5.3 Platform Agent

Health Platform Agents are largely outside the scope of this specification, but demonstration of how integration with a Marketplace instance connects with local container management is nevertheless useful. Limited example code showing connection and consumption of WebSocket events against the reference Marketplace Service is available at: https://github.com/hspc/hsp-agent.

6 APPENDIX A: MODEL QUICK REFERENCE

The following table provides a quick overview of all types supported by the Marketplace specification.

Type	Brief Description
IdentityProvider	Marketplace instance-level configuration parameters allowing user authentication against an external authority using OpenID Connect authentication flows.
User	A distinct person or system with some degree of access or interest to a Marketplace instance.
(User) Identity	IdentityProvider-specific information pertaining to a given User.
(User) Platform	Declaration of a compatible external Service runtime environment maintained by the User.
Group	A named collection of Users for purposes of batch Role assignment.
(Group) Member	Essentially a “join” record signifying a given User’s placement within a Group.
Role	A named set of permissions.
(Role) Appointment	The granting of a single Role to a single User or Group. (It is a polymorphic type.)
JsonWebToken	An RFC 7519 JSON Web Token issued to permit access by a given Identity to a Marketplace instance as a bearer token.
License	A known software or content license type, required to create Service records.
Service	Declaration of a Platform-compatible executable in the form of key metadata. Does not directly provide a reference to an executable image.
(Service) Screenshot	Optional graphical images for illustrating Service features to Users.
(Service) Build	Defines the reference to a specific versioned image of a given Service. Images must be hosted such that the Marketplace and its Users have read-only network access, at minimum.
(Service Build) Dependency	Known dependencies that are needed to run a given Build of a service.

(Service Build) Exposure	The standardized Interfaces capabilities provided by a given Service Build.
(Service Build Exposure) Parameter	States that configuration parameter of the given name is required at runtime to successfully provide the Interface of the Exposure.
(Service Build) Configuration	The runtime constraints of a Build that need to be known by a Platform for execution.
(Service Build Configuration) Task	A container entry point and associated constraints that must be run as part of a Configuration profile.
Interface	Marketplace-wide declaration of a standardized – or at least conventionalized – computational integration point. They are not constraints to HL7 standards.
(Interface) Surrogate	Marketplace-wide statement that the referenced substitute Interface provides compatible capabilities of the given base Interface. Useful for defining new versions of an Interface that are backwards compatible with older versions.

7 APPENDIX B: REFERENCED STANDARDS

Standard/Convention	See	Purpose
IETF SCIM 2	https://tools.ietf.org/html/rfc7644	User/Group provisioning
ISO 8601	https://www.iso.org/iso-8601-date-and-time-format.html	Date, time, and time zone standardization
OpenTracing	https://opentracing.io	SOA profiling and tracability
Open Container Initiative	https://www.opencontainers.org	Build containerization
OAuth 2	https://oauth.net/2/	Authorization
OpenID Connect	https://openid.net/connect/	Authentication
FHIR	http://www.fhir.org	Health services and data
SMART-on-FHIR	https://smarthealthit.org http://hl7.org/fhir/smart-app-launch/	Decoupled UIs
CDS Hooks	https://cds-hooks.org	Remote support services
HL7 CCOW	http://www.hl7.org/implement/standards/product_brief.cfm?product_id=1	User context synchronization