Microservices are an approach to distributed systems that promote the use of finely grained services with their own lifecycles, which collaborate together (from *O’Reilly Building Microservices)*. FPNG has adopted many of the properties of “textbook” Microservices. The primary reasons FPNG has moved to a Microservice architecture approach is:

a) Allow for each Microservice to have their own lifecycle (i.e.; deploy to production without worrying about the production release cycles of other FPNG Microservices); and

b) Reduce the time to build and run tests for only the Microservice that is changing.

Historically, FPNG was one monolithic application containing functionality related to many business domains (e.g., Shopping Cart (SC), Payment (PWP), Financial Manager (FM), Internal Admin Functionality (RDS), Maintenance Fee Storefront (MFS), and Inquiry Services (IS)). As a result it became hard to coordinate production release cycles. For example, if Shopping Cart was making a change that required 2 months of development, but MFS was making a change that required 4 months of development the Shopping Cart changes had to wait for the MFS changes to be completed before a production install could happen. It is also time consuming to run automated tests for the entire monolith if only one piece of functionality is changing. A Microservice approach allows only the relevant tests to run with confidence that “nothing else changed” because only that one Microservice will be deployed to production.

As FPNG development continues, new functionality will be evaluated by the FPNG System Architect to see if it warrants to be broken up into its own Microservice. Since the initial production install of the FPNG monolith in April 2016, the following Microservices have been created as new <major> functionality is being developed: FPNG-COLLECTIONS, FPNG-COMMON, FPNG-PRC-RLS-MGMT, FPNG-REFUND, FPNG-RQST-MGMT, FPNG-SECURITY-MGMT, FPNG-TSP. FPNG architects and management made the decision to not break apart the FPNG monolith (SC, MFS, PWP, RDS, IS, FM) until RAM is retired. In other words, only new functionality will be evaluated for a Microservice architecture until RAM is retired.

Having said that, there are some “textbook” attributes of Microservice architecture that FPNG has elected to not adhere to for technical, organizational and financial reasons. It is important to keep in mind that there isn’t a hard set of rules that govern how a system should be broken up into Microservices. To date, FPNG has chosen to break the system up into Microservices that are of moderate size – usually focused around a specific domain. As such, there has been some internal discussion about giving FPNG Microservices a different name – such as Macroservices. In an attempt to reduce confusion, below is a list of “textbook” attributes that FPNG’s Microservices follow and a list of attributes that FPNG does NOT follow.

**Textbook Attributes FPNG Microservices Follow:**

* Independent to deploy to production (without coordinating with other Microservicess)
* Easier to deploy
* Faster to test (without having to test other Microservices)
* Have their own source repository (Trunk)
* Have their own versioning
* Rely on “common” independently released code libraries
* “Small enough and no smaller” – while FPNG Microservices aren’t broken down by each POST/GET/PUT, and in certain communities this is the holy grail of Microservices, FPNG has broken the system into seems that can be managed by a small team.
* Communication between Microservices is over REST APIs
* Versioning of Services (e.g. v1, v2, v3)
* Each Microservice is built using its own pipeline (e.g., continuous integration & testing jobs)
* Lots of automated unit and service level testing
* Independent testing via mock/stub services
* Deployments to production are separated from release to customers via blue/green architecture
* Centralized logging aggregator (e.g,. Splunk)
* Common authentication mechanism (RBAC)
* Self-documenting services (Swagger)

**Textbook Attributes FPNG Microservices Do Not Follow:**

* Independent logging files (FPNG Microservices use the same server.log file)
* REST Services return UI content (FPNG has the UI as a separate war that is included within the Microservice)
* Dedicated VMs/Container per Microservice (FPNG Microservices share VMs)
* Scaling (no PaaS/Containers available to FPNG right now)
* Dedicated DB Schema per Microservice (FPNG has one shared DB Schema)
* Flipping from Blue/Green is done for the entire FPNG system – not per Microservice
* Monitoring, via AppD, is done for the entire FPNG system and not broken down by Microservice
* Static data is replicated across Microservices (lookup data)
* Automated E2E tests (this is done manually for FPNG)
* Correlation IDs (FPNG doesn’t currently have correlation IDs to map service calls by individual user)

### Setting up a FPNG Microservice

Follow the steps in the [“Microservice Task Template” document to setup a Microservice](https://epms.uspto.gov/PWA/Management%20Corporate%20Services/FPNG/Project%20Documents/3.%20Requirements/Technical%20Requirements/Environments/CICM/DevOps%20Phase%202/DevOps%20Phase%20II%20Process%20Documents/microservice%20task%20template.docx).