John de la Garza

john@jjdev.com - (714) 895 6311 - Westminster, CA 92683

Love programming and building systems. Have worked on things from microcontrollers to distributed systems and everything in between. Discovering how things work is something I enjoy and will have no trouble with any new hire learning curve.

SKILLS

- language: Python, Bash, C, Java, SQL, Go, Perl, Lua, JavaScript, Ruby, Lisp
- technology: HTTP, RDBMS (PostgreSQL, MySQL), DNS (BIND, Route53), Terraform, Linux (user and kernel), AWS (Lambda, ASG, ECS, ElastiCache (Redis), Kinesis, Security groups, etc), Nginx, OpenBSD, Tornado, Flask, Spring Boot, Elasticsearch

WORK EXPERIENCE

Blackberry, Irvine, CA, Senior Software Engineer Apr 2019 to Sep 2023

- Led design, feature improvements, and maintenance for our core systems which provided a REST API backend to a complex enterprise security product. Was responsible for a variety of tasks including developing new code, fixing bugs, deployments, sustaining, testing, and operations.
- Was one of the first responders when there was a customer-facing issue. This involved tracking down the cause in a distributed system with lots of moving parts and getting a fix deployed.
- Frequently pair programmed with others that were newer to the job to get them up to speed on using git consistently and learning how to run/develop/debug the different microservices.
- Based on the fact that our core services were I/O bound, an asynchronous Python3 framework called Tornado was used for many of the microservices. Asyncio in Python is based on coroutines that cooperatively yield control (instead of blocking) while waiting on blocking operations. This allows a single process to handle many requests concurrently. We had a backend of PostgreSQL (RDBMS), Redis (caching and pub/sub), and Elasticsearch.
- Was instrumental in transitioning several services from Python to Go.
- One of the more interesting things was the use of OpenResty/Lua (an application server that is forked from Nginx). This system had a listening socket on the Internet that customer devices would connect to with a web-socket, the endpoint would use this bidirectional socket to send and receive. Incoming messages were routed to a set of Kinesis streams where Lambdas would pick them up and process them. While outgoing messages originating in a web application (for example) would get sent to one of the HTTP microservices. This would result in a series of events that would ultimately publish the commands to a pub/sub message bus, where the OpenResty code would pick it up and send it down the socket.
- Async Python, Java, Lua, Go, .net, PostgreSQL, MySQL, AWS, Redis, Elasticsearch, DynamoDB

Comscore, Portland, OR, Senior Software Engineer Nov 2016 to Dec 2018

- Worked on a distributed system which was designed to count watched ads and determine who watched them. The system consisted of retrievers to download data from clients, importers to normalize and store the bulk of the data into a custom in-house parallel database, summarizers to store summary data into an RDBMS for use by the web front end, queuing subsystem, and exporters to transmit results back to clients.
- Created composable command line tools that acted as a language to assist in understanding incoming data and creating custom reports to be built combining these tools with some high level scripts.
- Participated in regular code reviews.
- C/C++, Perl, Python, RDBMS

Idealist.org, Portland, OR, Senior Developer — Jul 2015 to Jul 2016

- Worked on a team creating a REST API that needed to inter-operate with an existing model, database, and messaging system. It provided HTTPS endpoints to allow other systems to post jobs in an automated fashion. This involved doing a deep dive into the existing code to fully understand the existing architecture.
- Was involved in the creation of a new system that would allow customers to bulk post job listings. It was available to the end users as a REST API where they could submit jobs, check status, etc. It was consistent, well-documented, and designed to be discoverable.
- Wrote shell scripts to create a chroot based sandbox. This provided a reproducible development environment on individual developer computers.
- There was a lot of collaboration. Code reviews were almost always done. We frequently pair programmed when working on the more tricky parts. Work was done in two-week sprints.
- Python, PostgreSQL, AWS

Rally, Washington, DC, Operations Engineer — Jun 2014 to Mar 2015

- Implemented a configuration system that used HTTP to internally publish configuration files that lived in a git repository. Pushing to the configuration repository triggered the new files to be published. This service was deployed on AWS using Chef.
- Created a program that would run on a cron job. It would pull down XML data, process it, and insert it into a local database.
- Chef, Ruby, Python, AWS

TVplus, Orange, CA, Senior Developer — Aug 2012 to Jan 2014

- Designed and implemented a read/write caching layer for a REST API which was used by a JavaScript front end. Remote data was represented by a graph and the local data was a set of trees that could be lazily loaded.
- Designed and implemented a system to daily download XML data that represented TV program schedules for the next two weeks and merge/load them into a local database. This could mean scheduled shows were removed, added, or rescheduled. The system would try to re-attach content for shows that were rescheduled (if a match could be guessed) and flag all other cases of disassociated content for human intervention.
- Created a service that took a URL to a video file and a description that specified how to edit the video. The goal was to mute audio and replace the video with a black screen for the times and durations specified and then transcode to H.264/AAC.

  Once the new video file was created, a hash was generated and uploaded to an audio-matching service. A request was sent to another server to have it generate a different type of hash. Once this was done the path to the second hash was added to an audio-matching server.

  Most of this processing was done in parallel and utilized multiple cores well.
- Set up a development environment using LXC to allow a single server to run several systems to approximate our production environment which was spread across many Amazon EC2 nodes.
- Python, Tornado, Flask, PostgreSQL, AWS

Formal Education: Cal State Fullerton, BA Management Information Systems, Dean's List

Languages: English and Spanish