

DJANGO + CELERY

Do More with your Data and Less with your Code

“Torture the data, and it will confess to anything.”

—Ronald Coase

THE DATA PIPELINE



Each 6 phases of the Pipeline are ideally decoupled - independent services

For each step, make sure you are using the right tool for the job

The Goal is always to gain Insight

THE PROBLEMS

- I. Retrieving Data that is Not Yours
- II. Distributing the Workload on Data
- III. Managing when Data Processes are Run
- IV. Maintaining Data / Service Resiliency



ENTER TASK QUEUES

- I. Asynchronous Processing
- II. Decouples from a Specific Service
- III. Event Driven or Scheduled
- IV. Communication Between Systems (Messaging)
- V. Increased Data Resiliency



THE PROJECT

Let's build a Super Simple Message Service



- I. *Django + DRF* to expose RESTful Services to Clients
- II. *Celery* to handle Data Ingestion, Scheduling, and Distributed Processing
- III. *MySQL, MongoDB, and Redis* as our Data Storage

DJANGO + DRF

- I. Representational State Transfer (REST)
- II. Setup to productivity barrier is very low
- III. The Django Admin is Powerful
- IV. We like Python (a lot)
- V. Lots of great Community Support

CODE STRUCTURE

- I. Urls - Router, maps function/resource to browsable address
- II. Views - An exposed function; Akin to Controller in MVC
- III. Serializers - Data Validation and Formatting
- IV. Models - Data Definition, ORM Work Horse

CODE SNIPPETS

```
# Create a router and register our ViewSets

router = DefaultRouter()

router.register(r'messages', views.MessageViewSet)
router.register(r'images', views.ImageAttachmentViewSet)
router.register(r'users', views.UserViewSet)


urlpatterns = [
    url(r'^', include(router.urls)),
    path('celery/count_to_ten/<int:number_of_tasks>', views.CountToTen.as_view()),
    path('celery/scrape_image', views.ScrapeImage.as_view())
]
```

CODE SNIPPETS

```
class MessageViewSet(viewsets.ModelViewSet):  
    """  
    Provides List, Detail, Retrieve, Create, Update, Delete for Model  
    """  
  
    queryset = Message.objects.all()  
    serializer_class = MessageSerializer  
    permission_classes = (permissions.IsAuthenticated, )  
  
    def perform_create(self, serializer):  
        serializer.save(author=self.request.user)
```

CODE SNIPPETS

```
class MessageSerializer(serializers.HyperlinkedModelSerializer):  
    author = serializers.ReadOnlyField(source='author.username')  
    image_attachment = serializers.HyperlinkedRelatedField(view_name='image-detail',  
                                                            many=True,  
                                                            read_only=True)  
  
    class Meta:  
        model = Message  
        fields = '__all__'
```


CODE SNIPPETS

```
class Message(models.Model):  
    """  
    A message from one user to another  
    """  
  
    subject = models.CharField(max_length=50, blank=True, default='')  
    body = models.CharField(max_length=2000, blank=True, default='')  
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='messages')  
    to_user = models.ForeignKey(User, on_delete=models.DO_NOTHING, null=True)  
    active = models.BooleanField(default=True)  
    modified_date = models.DateTimeField(auto_now_add=True)  
    created_date = models.DateTimeField(auto_now_add=True)  
  
    def __str__(self):  
        return self.subject  
  
    class Meta:  
        managed = True  
        ordering = ('created_date', )  
        db_table = 'message'  
        verbose_name = 'User Message'  
        verbose_name_plural = 'User Messages'
```

CELERY

- I. Distributed Asynchronous Task Management
- II. Decoupled Self-Monitoring Processes
- III. Task Scheduling (Celery-Beat)
- IV. Feature Rich
- V. Well Supported Community

CODE STRUCTURE

- I. Views - Exposed Service
- II. Tasks - Defines the High Level Processes
- III. Queues - A list of ordered Tasks for Workers
- IV. Workers - Subscribe to a Queue, the Work Horse

CODE SNIPPETS

```
class ScrapeImage(APIView):  
    """  
    Dispatches a task for a given image url.  
    """  
  
    def get(self, request):  
        image_url = request.GET.get('image_url')  
        scrape_image.delay(user=request.user.pk, image_url=image_url)  
        return Response(f'Dispatched a task to grab the image at {image_url}.')
```

CODE SNIPPETS

```
@task
def scrape_image(user, image_url):
    print(f'Grabbing the image at: {image_url}')
    sleep(5)
    # Get the user
    user_obj = User.objects.get(pk=user)
    Image.objects.create(owner=user_obj, url=image_url)
    print('Finished grabbing and storing image in the db.')
```

CODE SNIPPETS

```
app = Celery()

@app.on_after_configure.connect
def setup_periodic_tasks(user, image_url):
    # Executes every Monday morning at 8:30 a.m.
    sender.add_periodic_task(
        crontab(hour=8, minute=30, day_of_week=1),
        scrape_image(user, url),
    )
```


DEMO

CONCLUSIONS

- I. Modern Systems ask *a lot* of their Data at almost every layer, so be prepared to handle it.
- II. Decouple Services from Data Processes
- III. Django + Celery is awesome because it is Powerful
- IV. Django + Celery is awesome because it is Simple



ADDENDUM

- I. kombu-fernet-serializers from Heroku to add Security
- II. Celery supports a wide range of Brokers, Results Stores, and Data Serialization Tools
- III. Celery has integrations with other popular web frameworks
- IV. We got through this without a *single* Django Unchained joke (maybe)...

CREDITS

Presenters

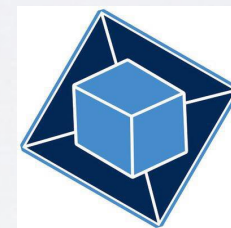
Elliott V. Iannello

Software Developer
elliott.iannello@autoipacket.com

Brandon M. Phillips

Software Engineer
brandon.phillips@autoipacket.com

Sponsors



Morgantown Codes

Special Thanks to
Darin Markus and Kel Cecil
for helping organize this Event :)

THANKS FOR LISTENING

—

QUESTIONS?