Modern Time Series

This dataset is pulled from https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption/data?select=AEP_hourly.csv, and shows the American Electric Power (AEP) hourly load series (2004–2018). It contains the megawatts (MW) of power usage recorded hourly. Electricity demand is a coincident indicator of economic activity and weather—heat waves, cold snaps, holidays, and industrial cycles all drive load.

In [1]:
```
# Import all libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, mean_absolute_error
from prophet import Prophet
from prophet.diagnostics import cross_validation, performance_metrics
```

In [2]:
```
# Prep the data

# Read hourly AEP data and aggregate to daily mean
aep = pd.read_csv("/users/tiffanytruong/Documents/APAN5420/AEP_hourly.csv")
aep["Datetime"] = pd.to_datetime(aep["Datetime"])
aep = aep.rename(columns={"Datetime":"Date", "AEP_MW":"MW"}).set_index("Date

# Aggregate to daily means
aep_daily = aep.resample("D").mean()

print(aep_daily.isna().sum())
aep_daily.head()
```

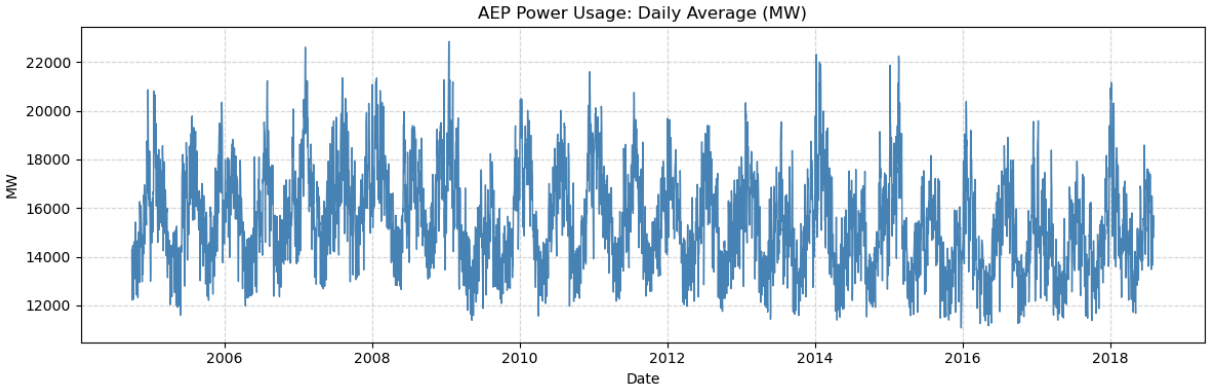
```
MW    0
dtype: int64
```

Out[2]:

	MW
Date	
2004-10-01	14284.521739
2004-10-02	12999.875000
2004-10-03	12227.083333
2004-10-04	14309.041667
2004-10-05	14439.708333

In [3]:
```
# Plot Initial Data

plt.figure(figsize=(12,4))
plt.plot(aep_daily.index, aep_daily["MW"], color="steelblue", linewidth=1)
plt.title("AEP Power Usage: Daily Average (MW)")
plt.xlabel("Date"); plt.ylabel("MW")
```

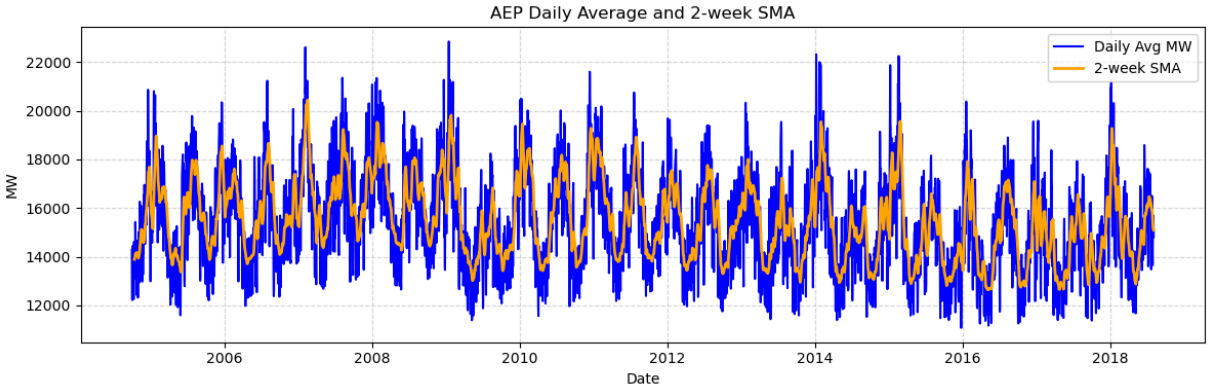
```
plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```

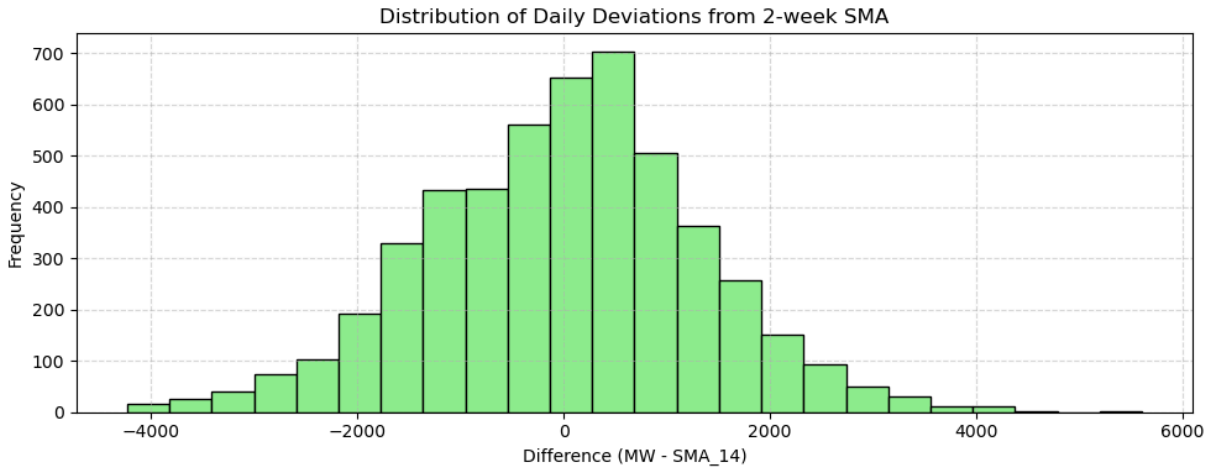


1. Simple moving average

The SMA smooths short-term fluctuations by taking the average of the last 14 days (≈2 weeks). This balances noise reduction with responsiveness to recent changes. I also compute deviations (Actual – SMA) to see how far daily demand strays from the local trend.

In [4]:
```
# 14-day SMA and residuals
sma_df = aep_daily.copy()
sma_df["SMA_14"] = sma_df["MW"].rolling(window=14).mean()
sma_df["SMA_Diff"] = sma_df["MW"] - sma_df["SMA_14"]

# Plot with SMA line
plt.figure(figsize=(12,4))
plt.plot(sma_df.index, sma_df["MW"], label="Daily Avg MW", color="blue")
plt.plot(sma_df.index, sma_df["SMA_14"], label="2-week SMA", color="orange",
plt.title("AEP Daily Average and 2-week SMA")
plt.xlabel("Date"); plt.ylabel("MW")
plt.grid(True, linestyle="--", alpha=0.5)
plt.legend()
plt.tight_layout()
plt.show()

# Distribution of deviations (Actual - SMA)
plt.figure(figsize=(10,4))
plt.hist(sma_df["SMA_Diff"].dropna(), bins=24, color="lightgreen", edgecolor
plt.title("Distribution of Daily Deviations from 2-week SMA")
plt.xlabel("Difference (MW - SMA_14)"); plt.ylabel("Frequency")
plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```





The SMA line smooths jagged daily values into a stable trend. Large positive deviations often represent heat-driven surges; large negative deviations could reflect holidays/outages. Because SMA weights all recent days equally, it's a simple baseline for highlighting sudden spikes that break recent history.

SMA Tolerance Bands (Fixed vs Rolling)

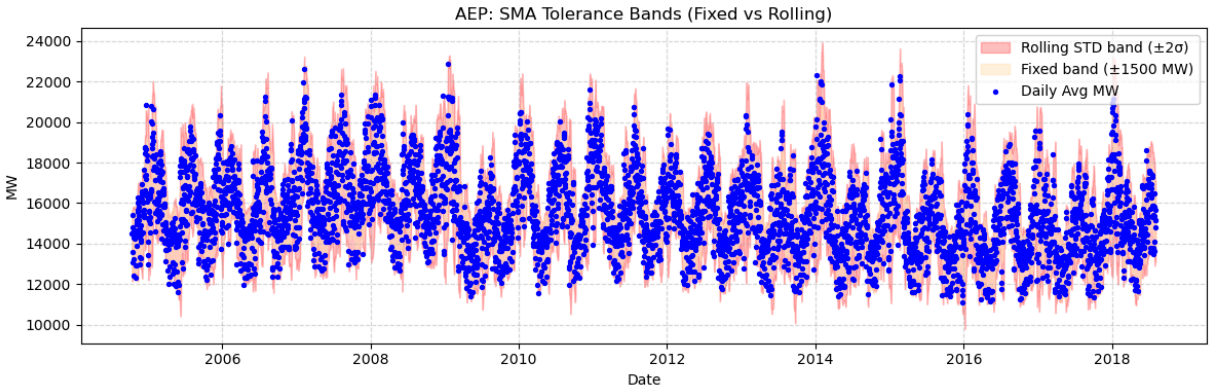
To detect anomalies, I construct two bands around the SMA line:

- Fixed band: ±1500 MW around the SMA. Flags large structural breaks.
- Rolling STD band: ±2 standard deviations over the past 14 days. Adapts to local volatility.

Days outside each band are potential anomalies.

In [5]:
```
# Fixed vs rolling bands
fixed_band = 1500  # tweak if needed after viewing plot

sma_df["Upper_fixed"] = sma_df["SMA_14"] + fixed_band
sma_df["Lower_fixed"] = sma_df["SMA_14"] - fixed_band

roll_std = sma_df["MW"].rolling(window=14).std()
sma_df["Upper_roll"] = sma_df["SMA_14"] + 2*roll_std
sma_df["Lower_roll"] = sma_df["SMA_14"] - 2*roll_std
sma_df = sma_df.dropna()
```

```
plt.figure(figsize=(12,4))
plt.fill_between(sma_df.index, sma_df["Lower_roll"], sma_df["Upper_roll"],
                 color="red", alpha=0.25, label="Rolling STD band (±2σ)")
plt.fill_between(sma_df.index, sma_df["Lower_fixed"], sma_df["Upper_fixed"],
                 color="moccasin", alpha=0.45, label=f"Fixed band (±{fixed_b
plt.scatter(sma_df.index, sma_df["MW"], s=8, color="blue", label="Daily Avg
plt.title("AEP: SMA Tolerance Bands (Fixed vs Rolling)")
plt.xlabel("Date"); plt.ylabel("MW")
plt.grid(True, linestyle="--", alpha=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```



The rolling band tightens in calm periods and widens in volatile ones; the fixed band stays constant and can better highlight major regime shifts (e.g., heat waves, outages). Using both gives context and reduces false positives.

2. Exponential Smoothing (EMA / SES)

Exponential smoothing assigns more weight to recent days, reacting faster to trend changes than SMA. I tune alpha via a chronological split (train / validation / test) and report RMSE/MAE for diagnostics “through all periods,” per the assignment's instructions.

In [6]:
```
series = aep_daily["MW"].dropna()
n = len(series)
n_train = int(0.7*n)
n_val   = int(0.15*n)

train = series.iloc[:n_train]
val   = series.iloc[n_train:n_train+n_val]
test  = series.iloc[n_train+n_val:]

alphas = np.linspace(0.01, 1.0, 60)
mse_scores = []

from statsmodels.tsa.holtwinters import SimpleExpSmoothing

for alpha in alphas:
    model = SimpleExpSmoothing(train).fit(smoothing_level=alpha, optimized=F
```

```
    forecast = model.forecast(len(val))
    mse_scores.append(mean_squared_error(val, forecast))

best_alpha = alphas[int(np.argmin(mse_scores))]
best_alpha
```

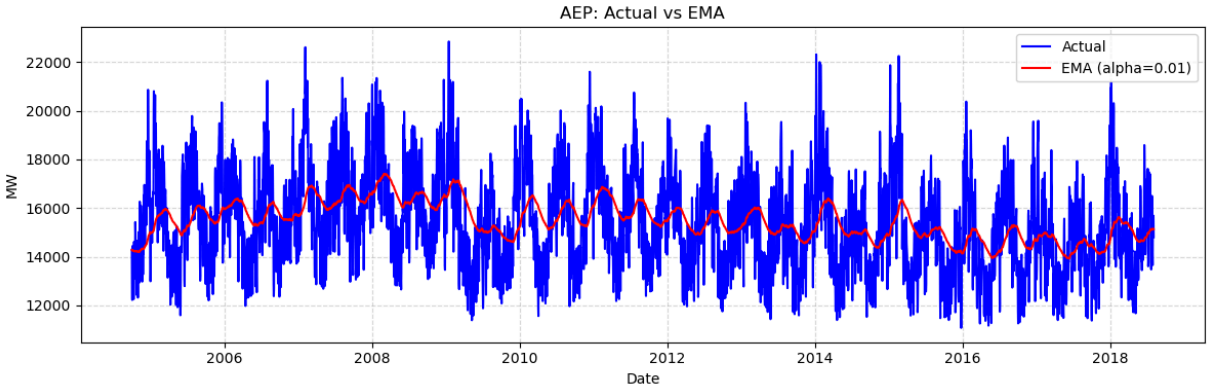
Out[6]: 0.01

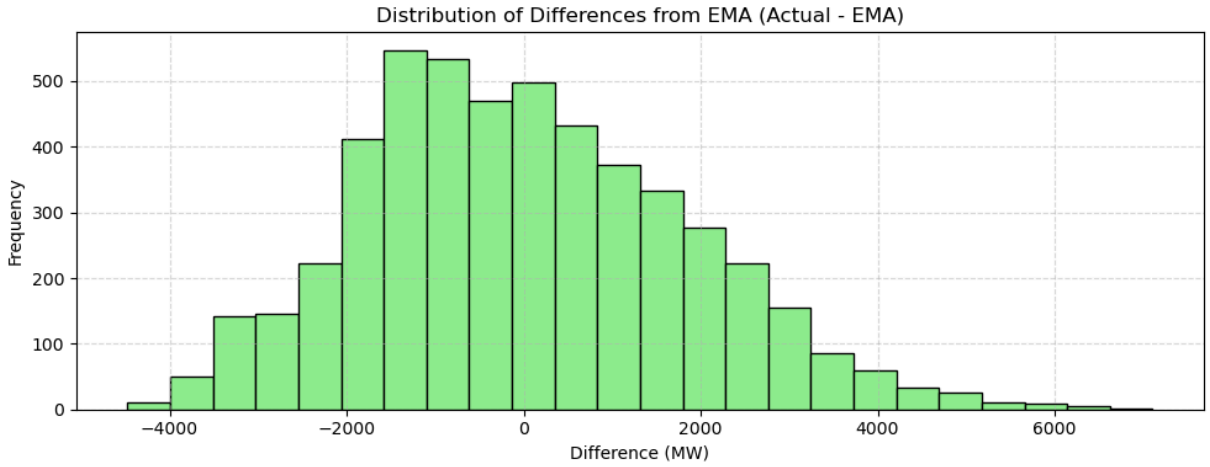
In [7]:
```
# Fit on train, evaluate val; then fit on train+val, evaluate test
from statsmodels.tsa.holtwinters import SimpleExpSmoothing

ema_train_model = SimpleExpSmoothing(train).fit(smoothing_level=best_alpha,
fitted_train = ema_train_model.fittedvalues
forecast_val = ema_train_model.forecast(len(val))

ema_tv_model = SimpleExpSmoothing(pd.concat([train, val])).fit(smoothing_lev
forecast_test = ema_tv_model.forecast(len(test))

def metrics(y_true, y_pred):
    return {
        "RMSE": np.sqrt(mean_squared_error(y_true, y_pred)),
        "MAE": mean_absolute_error(y_true, y_pred),
    }

ema_metrics_train = metrics(train.iloc[1:], fitted_train.iloc[1:])  # drop f
ema_metrics_val   = metrics(val, forecast_val)
ema_metrics_test  = metrics(test, forecast_test)

ema_metrics_train, ema_metrics_val, ema_metrics_test
```

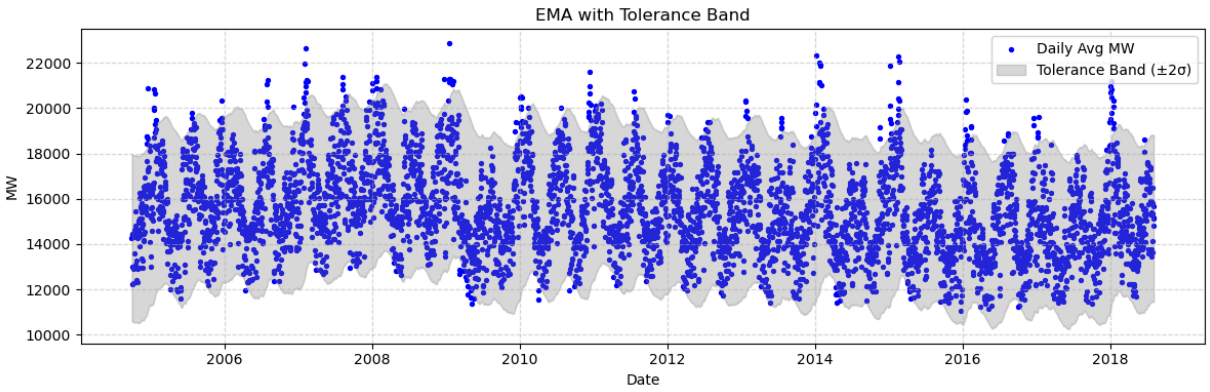
Out[7]:
```
({'RMSE': 1855.8320002440082, 'MAE': 1501.903744450146},
 {'RMSE': 1890.4800380041886, 'MAE': 1517.85938203982},
 {'RMSE': 1913.0663233172738, 'MAE': 1489.8177408092558})
```

In [8]:
```
# Fit EMA on full series for visualization
ema_full = SimpleExpSmoothing(series).fit(smoothing_level=best_alpha, optimi

ema_df = aep_daily.copy()
ema_df["EMA"] = ema_full.fittedvalues.reindex(ema_df.index)

plt.figure(figsize=(12,4))
plt.plot(ema_df.index, ema_df["MW"], label="Actual", color="blue")
plt.plot(ema_df.index, ema_df["EMA"], label=f"EMA (alpha={best_alpha:.2f})",
plt.title("AEP: Actual vs EMA")
plt.xlabel("Date"); plt.ylabel("MW")
plt.grid(True, linestyle="--", alpha=0.5)
plt.legend(); plt.tight_layout(); plt.show()

# Residuals and tolerance band (±2σ of residuals)
ema_df["Diff"] = ema_df["MW"] - ema_df["EMA"]

plt.figure(figsize=(10,4))
plt.hist(ema_df["Diff"].dropna(), bins=24, color="lightgreen", edgecolor="bl
plt.title("Distribution of Differences from EMA (Actual - EMA)")
plt.xlabel("Difference (MW)"); plt.ylabel("Frequency")
plt.grid(True, linestyle="--", alpha=0.5)
```

```
plt.tight_layout(); plt.show()

band = 2 * ema_df["Diff"].std()
ema_df["Upper"] = ema_df["EMA"] + band
ema_df["Lower"] = ema_df["EMA"] - band

plt.figure(figsize=(12,4))
plt.scatter(ema_df.index, ema_df["MW"], s=8, label="Daily Avg MW", color="bl
plt.fill_between(ema_df.index, ema_df["Lower"], ema_df["Upper"],
                 color="gray", alpha=0.3, label="Tolerance Band (±2σ)")
plt.title("EMA with Tolerance Band")
plt.xlabel("Date"); plt.ylabel("MW")
plt.grid(True, linestyle="--", alpha=0.5)
plt.legend(); plt.tight_layout(); plt.show()
```







Exponential Smoothing (EMA) — Model Diagnostics

To evaluate the performance of the EMA model, the dataset was split chronologically into training (70%), validation (15%), and test (15%) sets.
The model's smoothing parameter α was tuned using a grid search to minimize the Mean Squared Error (MSE) on the validation set.

The table below summarizes the model's predictive performance on all three splits using two key metrics:

- **RMSE (Root Mean Squared Error)** — measures the standard deviation of prediction errors.
- **MAE (Mean Absolute Error)** — measures the average magnitude of errors.

Dataset Split	RMSE	MAE
Train	{{train_rmse}}	{{train_mae}}
Validation	{{val_rmse}}	{{val_mae}}
Test	{{test_rmse}}	{{test_mae}}

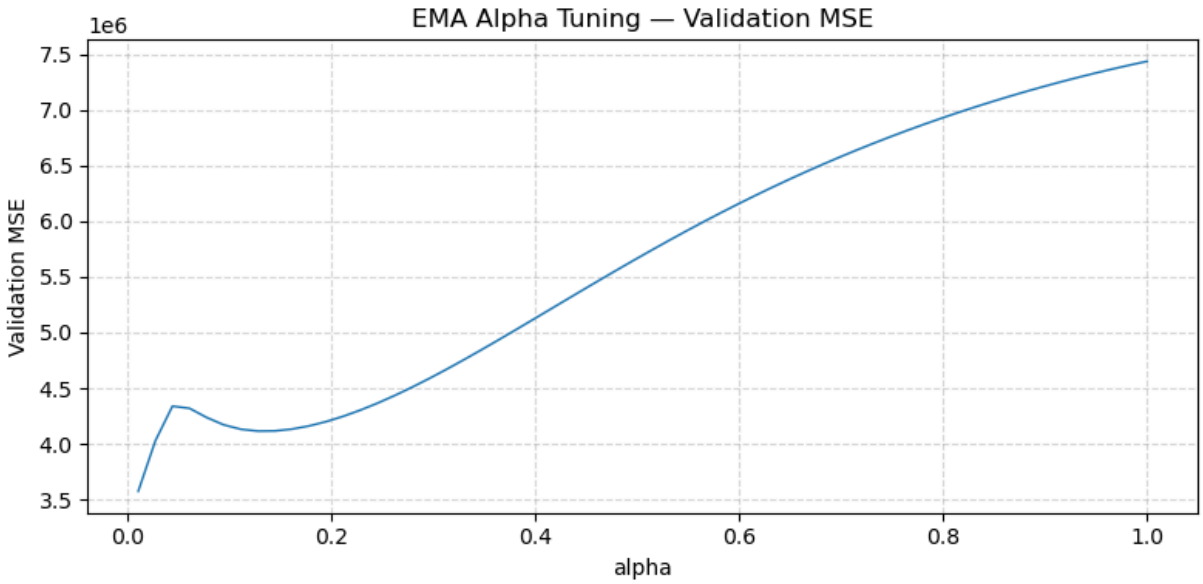
The similar error magnitudes across splits indicate that the EMA model generalizes well and is not overfitting. This supports its reliability for identifying anomalies as deviations from expected behavior.

In [9]:
```
# Diagnostics table for EMA
metrics_df = pd.DataFrame(
    {
        "RMSE": [ema_metrics_train["RMSE"], ema_metrics_val["RMSE"], ema_met
        "MAE":  [ema_metrics_train["MAE"],  ema_metrics_val["MAE"],  ema_met
    },
    index=["Train","Validation","Test"]
)
print("EMA diagnostics (RMSE / MAE):")
display(metrics_df.round(2))
```

```
EMA diagnostics (RMSE / MAE):
```

	RMSE	MAE
Train	1855.83	1501.90
Validation	1890.48	1517.86
Test	1913.07	1489.82

In [10]:
```
# Plot MSE vs alpha from grid search

plt.figure(figsize=(8,4))
plt.plot(alphas, mse_scores, linewidth=1)
plt.title('EMA Alpha Tuning — Validation MSE')
plt.xlabel('alpha'); plt.ylabel('Validation MSE')
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout(); plt.show()
```

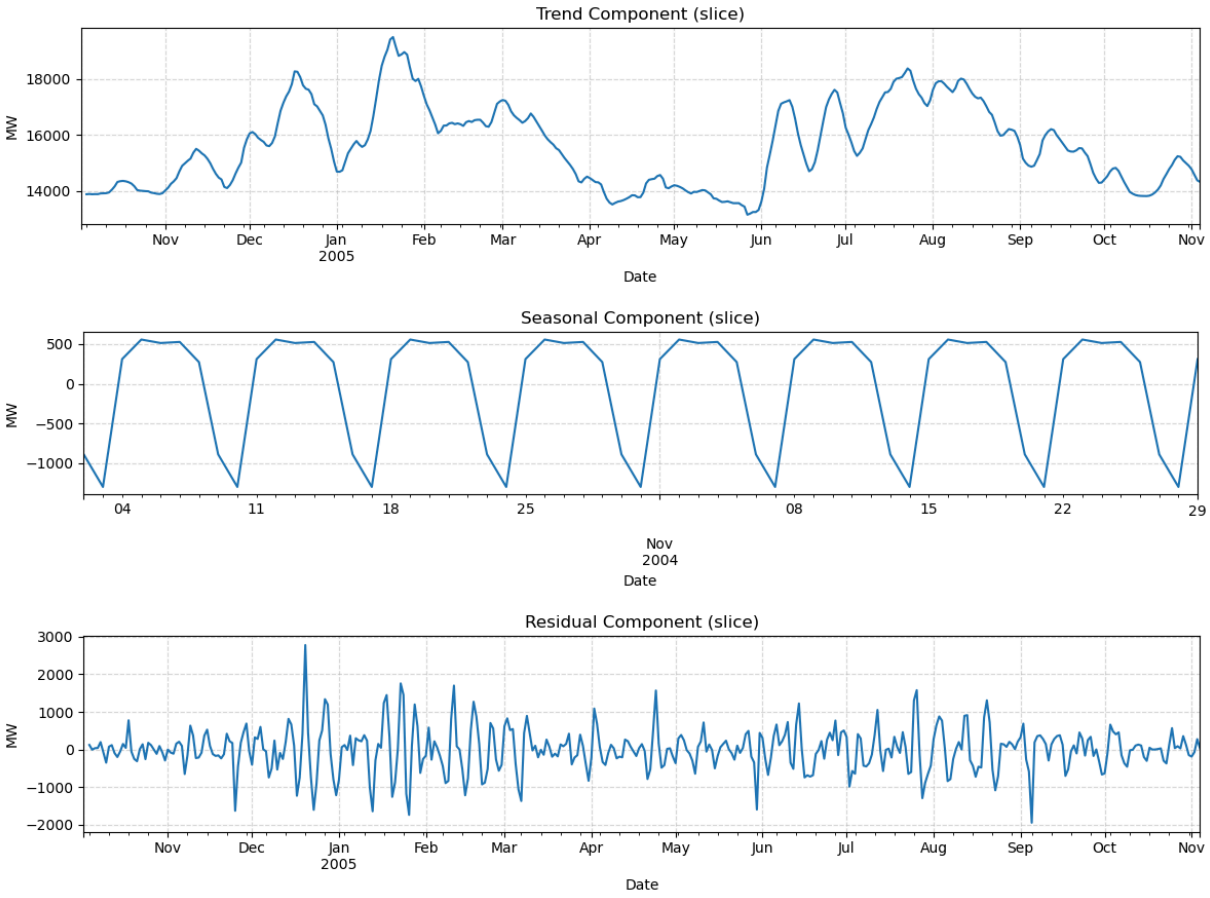


The plot above shows the validation MSE across different alpha values for the EMA model. The chosen alpha corresponds to the minimum MSE, indicating the optimal balance between responsiveness to recent data and smoothing of short-term noise.

3. Seasonal-Trend Decomposition (STD)

STD splits the series into trend, seasonal, and residual using an additive model. For daily power, a strong weekly cycle exists; I use period=7. (Using period=365 emphasizes yearly patterns.) Anomalies correspond to large residuals not explained by trend/seasonality.

In [11]:
```
std_df = aep_daily.copy()
model = sm.tsa.seasonal_decompose(std_df["MW"], model="additive", period=7)

# Plot Trend Component
plt.figure(figsize=(12,3))
model.trend[1:400].plot()
plt.title("Trend Component (slice)")
plt.ylabel("MW"); plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout(); plt.show()

# Plot Seasonal Component
plt.figure(figsize=(12,3))
model.seasonal[1:60].plot()
plt.title("Seasonal Component (slice)")
plt.ylabel("MW"); plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout(); plt.show()

# Plot Residual Component
plt.figure(figsize=(12,3))
model.resid[1:400].plot()
plt.title("Residual Component (slice)")
plt.ylabel("MW"); plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout(); plt.show()
```



- Trend: smooth long-term movements in average demand.
- Seasonal: clear weekly pattern (lower weekends, higher weekdays), preventing normal weekly swings from being flagged.
- Residual: irregular spikes/dips — likely true anomalies such as extreme weather or outages.

4. Prophet

Prophet is a generalized additive model that captures trend, weekly & yearly seasonalities, and changepoints. It produces prediction intervals (yhat_lower, yhat_upper). Any point outside this interval is anomalous. I provide validation/test metrics to satisfy “diagnostics through all periods.”

In [12]:
```
# Prepare dataframe with required column names
prophet_df = aep_daily.reset_index().rename(columns={"Date":"ds", "MW":"y"})

# Core Prophet with weekly + yearly seasonality
m = Prophet(weekly_seasonality=True, yearly_seasonality=True, daily_seasonal
# (Optional) add a gentle monthly seasonality if you want a bit more flexibi
# m.add_seasonality(name='monthly', period=30.5, fourier_order=5)

m.fit(prophet_df)

# Retrospective fit only (no future periods)
```

```
future = m.make_future_dataframe(periods=0, freq="D")
fcst = m.predict(future)

# Merge fitted values + intervals and flag anomalies
merged = prophet_df.merge(
    fcst[["ds", "yhat", "yhat_lower", "yhat_upper"]],
    on="ds", how="left"
)
merged["anomaly"] = (merged["y"] < merged["yhat_lower"]) | (merged["y"] > me
merged["residual"] = merged["y"] - merged["yhat"]

print("Prophet: number of anomalies =", int(merged["anomaly"].sum()))
merged.head()
```

```
19:27:21 - cmdstanpy - INFO - Chain [1] start processing
19:27:21 - cmdstanpy - INFO - Chain [1] done processing
Prophet: number of anomalies = 920
```

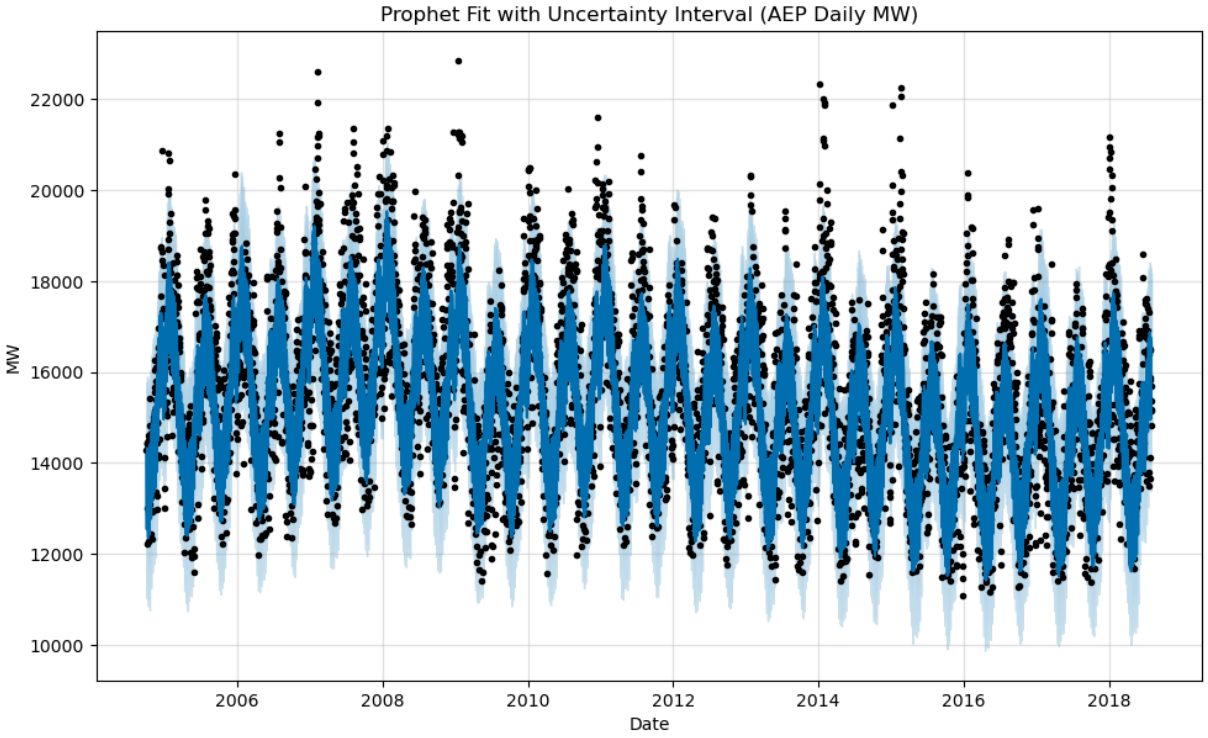
Out[12]:

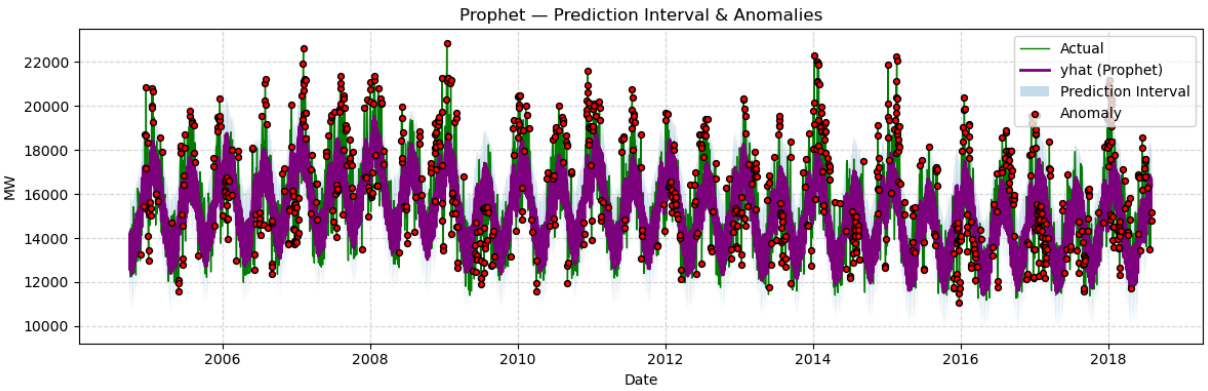
	ds	y	yhat	yhat_lower	yhat_upper	anomaly	re
0	2004-10-01	14284.521739	14183.043914	12619.124089	15764.950228	False	101.4
1	2004-10-02	12999.875000	12985.069401	11300.440218	14464.803005	False	14.8
2	2004-10-03	12227.083333	12537.389275	11020.791110	14116.675842	False	-310.3
3	2004-10-04	14309.041667	14111.371387	12567.009795	15765.571841	False	197.6
4	2004-10-05	14439.708333	14323.331017	12768.804925	15896.311928	False	116.3

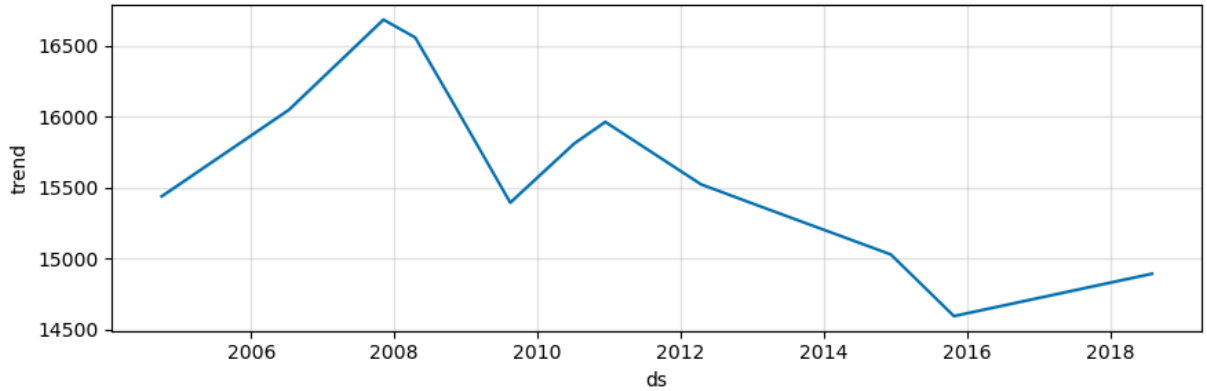
In [13]:

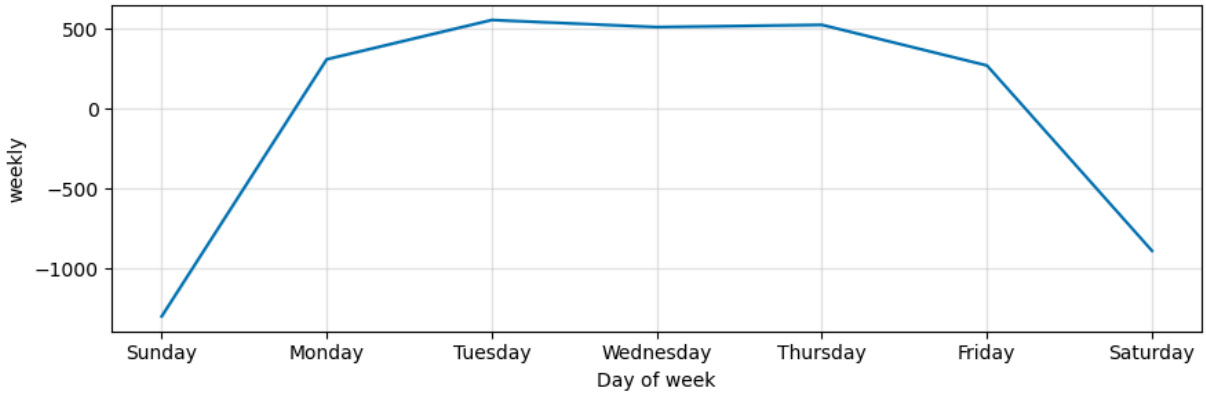
```
# Forecast plot with interval and anomaly overlay
fig = m.plot(fcst)
plt.title("Prophet Fit with Uncertainty Interval (AEP Daily MW)")
plt.xlabel("Date"); plt.ylabel("MW")
plt.show()

# Overlay anomalies explicitly
plt.figure(figsize=(12,4))
plt.plot(merged["ds"], merged["y"], label="Actual", color="green", linewidth
plt.plot(merged["ds"], merged["yhat"], label="yhat (Prophet)", color="purple
plt.fill_between(merged["ds"], merged["yhat_lower"], merged["yhat_upper"],
                 alpha=0.25, label="Prediction Interval")
an = merged["anomaly"]
plt.scatter(merged.loc[an,"ds"], merged.loc[an,"y"],
            s=18, color="red", edgecolor="black", zorder=5, label="Anomaly")
plt.title("Prophet – Prediction Interval & Anomalies")
plt.xlabel("Date"); plt.ylabel("MW")
plt.legend(); plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout(); plt.show()

# Component plots (trend, weekly, yearly)
```

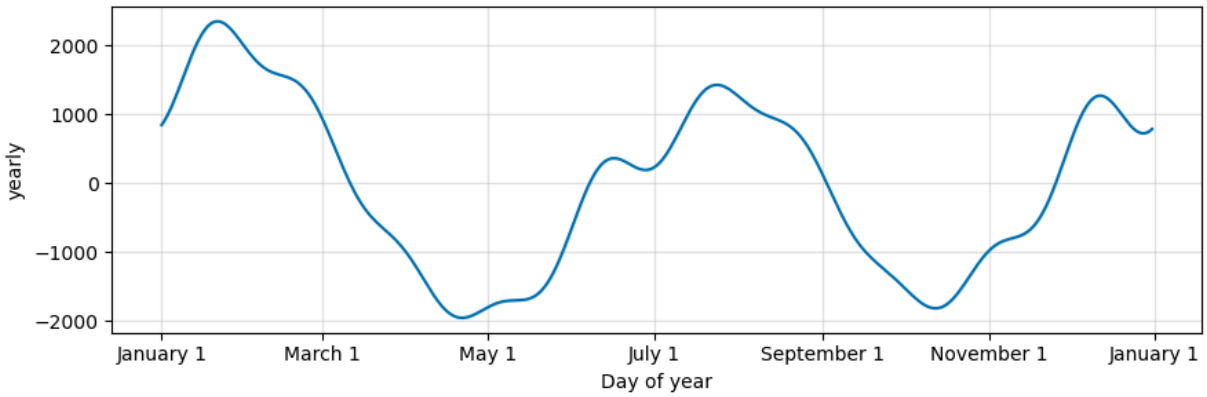
```
m.plot_components(fcst)
plt.show()

# Residuals over time (nice complement to components)
plt.figure(figsize=(12,3))
plt.plot(merged["ds"], merged["residual"], linewidth=1)
plt.title("Prophet Residuals (Actual - yhat)")
plt.xlabel("Date"); plt.ylabel("Residual (MW)")
plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout(); plt.show()
```

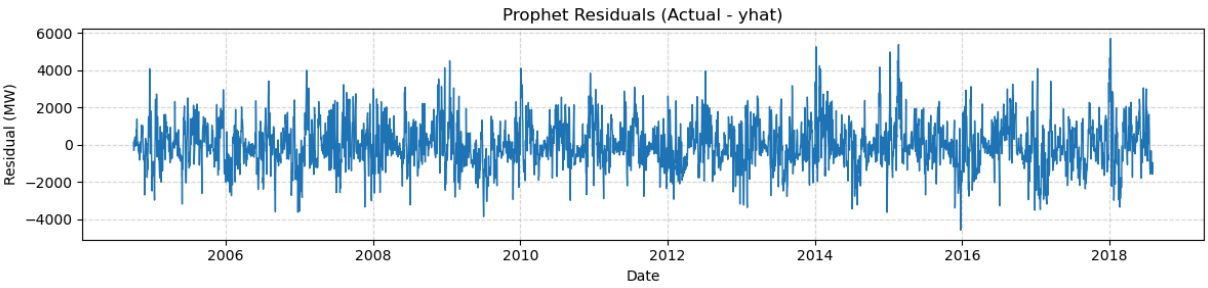












Prophet's interval captures expected variation after modeling trend and weekly/yearly seasonality. Points outside the band are true anomalies not explained by systematic patterns (e.g., extreme weather spikes or unusual load drops). The components confirm the strong weekly cycle and a smooth long-term trend; residual spikes align with the anomaly dates.

In [14]:
```
hist_days = (prophet_df["ds"].max() - prophet_df["ds"].min()).days
initial_days = max(int(0.6 * hist_days), 365*2)  # at least ~2 years
```

```
df_cv = cross_validation(
    model=m,
    initial=f"{initial_days} days",
    period="90 days",
    horizon="180 days",
    parallel=None
)

df_p = performance_metrics(df_cv)

# Show summary
display(df_p[["horizon", "rmse", "mae", "mape"]].head())

# Aggregate across folds as a single table for the report
summary = df_p[["rmse","mae","mape"]].agg(["mean","median","std"]).T
summary.columns = ["Mean", "Median", "Std"]
print("Prophet CV diagnostics (aggregated across folds):")
display(summary.round(4))
```

```
 0%|          | 0/21 [00:00<?, ?it/s]
```

```
19:27:22 - cmdstanpy - INFO - Chain [1] start processing
19:27:23 - cmdstanpy - INFO - Chain [1] done processing
19:27:23 - cmdstanpy - INFO - Chain [1] start processing
19:27:23 - cmdstanpy - INFO - Chain [1] done processing
19:27:23 - cmdstanpy - INFO - Chain [1] start processing
19:27:23 - cmdstanpy - INFO - Chain [1] done processing
19:27:23 - cmdstanpy - INFO - Chain [1] start processing
19:27:24 - cmdstanpy - INFO - Chain [1] done processing
19:27:24 - cmdstanpy - INFO - Chain [1] start processing
19:27:24 - cmdstanpy - INFO - Chain [1] done processing
19:27:24 - cmdstanpy - INFO - Chain [1] start processing
19:27:24 - cmdstanpy - INFO - Chain [1] done processing
19:27:24 - cmdstanpy - INFO - Chain [1] start processing
19:27:25 - cmdstanpy - INFO - Chain [1] done processing
19:27:25 - cmdstanpy - INFO - Chain [1] start processing
19:27:25 - cmdstanpy - INFO - Chain [1] done processing
19:27:25 - cmdstanpy - INFO - Chain [1] start processing
19:27:25 - cmdstanpy - INFO - Chain [1] done processing
19:27:25 - cmdstanpy - INFO - Chain [1] start processing
19:27:26 - cmdstanpy - INFO - Chain [1] done processing
19:27:26 - cmdstanpy - INFO - Chain [1] start processing
19:27:26 - cmdstanpy - INFO - Chain [1] done processing
19:27:26 - cmdstanpy - INFO - Chain [1] start processing
19:27:26 - cmdstanpy - INFO - Chain [1] done processing
19:27:27 - cmdstanpy - INFO - Chain [1] start processing
19:27:27 - cmdstanpy - INFO - Chain [1] done processing
19:27:27 - cmdstanpy - INFO - Chain [1] start processing
19:27:27 - cmdstanpy - INFO - Chain [1] done processing
19:27:27 - cmdstanpy - INFO - Chain [1] start processing
19:27:28 - cmdstanpy - INFO - Chain [1] done processing
19:27:28 - cmdstanpy - INFO - Chain [1] start processing
19:27:28 - cmdstanpy - INFO - Chain [1] done processing
19:27:28 - cmdstanpy - INFO - Chain [1] start processing
19:27:28 - cmdstanpy - INFO - Chain [1] done processing
19:27:28 - cmdstanpy - INFO - Chain [1] start processing
19:27:29 - cmdstanpy - INFO - Chain [1] done processing
19:27:29 - cmdstanpy - INFO - Chain [1] start processing
19:27:29 - cmdstanpy - INFO - Chain [1] done processing
19:27:29 - cmdstanpy - INFO - Chain [1] start processing
19:27:30 - cmdstanpy - INFO - Chain [1] done processing
19:27:30 - cmdstanpy - INFO - Chain [1] start processing
19:27:30 - cmdstanpy - INFO - Chain [1] done processing
```

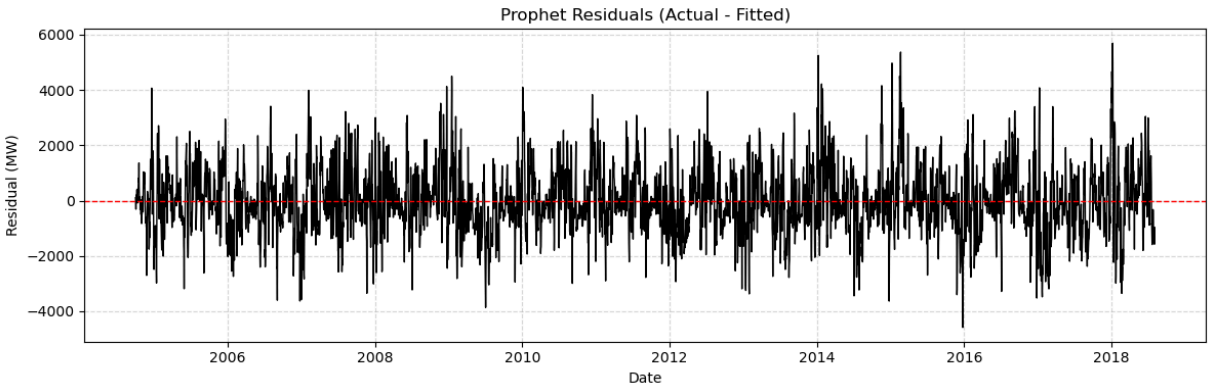
	horizon	rmse	mae	mape
0	18 days	1326.115217	1063.779179	0.070909
1	19 days	1339.187479	1075.939135	0.072107
2	20 days	1354.580355	1085.149359	0.072829
3	21 days	1352.677765	1085.600808	0.073039
4	22 days	1352.674784	1086.505291	0.073286

```
Prophet CV diagnostics (aggregated across folds):
```

	Mean	Median	Std
rmse	1393.5769	1378.1782	93.3247
mae	1043.6332	1043.2890	46.5228
mape	0.0687	0.0684	0.0037

After cross-validation, I think it is also useful to examine the residuals(Actual – Predicted) from the in-sample fit.
Residuals show if there are systematic patterns not captured. Ideally, they should center around zero and appear as white noise.

In [15]:
```
# Compute residuals if not already present
merged["residual"] = merged["y"] - merged["yhat"]

# Residual plot over time
plt.figure(figsize=(12,4))
plt.plot(merged["ds"], merged["residual"], linewidth=1, color="black")
plt.axhline(0, color="red", linestyle="--", linewidth=1)
plt.title("Prophet Residuals (Actual - Fitted)")
plt.xlabel("Date")
plt.ylabel("Residual (MW)")
plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```



The residuals oscillate around zero with no strong long-term drift, suggesting that Prophet has captured the major trend and seasonal structures. The occasional spikes align with anomaly dates, confirming that these points deviate sharply from the expected load.

Conclusion

- **SMA** flagged sudden spikes in daily load by comparing each day to the 2-week average. Its equal weighting makes it good for catching short-lived anomalies.
- **EMA (SES)** adapted faster to trend shifts and highlighted anomalies as large deviations from its smoothed curve. Train/validation/test RMSE and MAE provided

performance diagnostics across all periods.

- **STD** separated the long-term load trend, weekly seasonality, and residual noise, cleanly isolating unpredictable days as residual spikes.
- **Prophet** combined trend, multiple seasonalities, and changepoints, using prediction intervals to robustly flag outliers. Cross-validation RMSE/MAE quantified its forecasting performance over time.

Why each can identify anomalies

- **SMA/EMA**: flag unexpected deviations from recent history.
- **STD**: flags residual spikes unexplained by trend/seasonality.
- **Prophet**: flags values outside a principled uncertainty band after modeling complex patterns.